Broadview®
www.broadview.com.cn

自己动手做大数据系统

张魁 张粤磊 刘未昕 吴茂贵 / 著

中国工信出版集团
電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

· 张 魁 ·

虚拟化工程师，OpenStack架构师，苏州某高校云平台架构师，十余年Linux系统运维实践及虚拟化开发经验，4年Linux系统补丁开发经验。先后在美企担任虚拟化应用运维、服务器集群开发运维工程师或系统开发架构师，高校信息中心云平台架构师，主要关注OpenStack、Docker及分布式存储等。

· 张粤磊 ·

DBA、大数据架构师，十余年一线数据处理数据分析实战经验。先后在咨询、金融、互联网行业担任数据平台技术负责人或架构师。主要关注大数据基础平台、大数据模型构建和大数据分析。

自己动手做大数据系统

张魁 张粤磊 刘未昕 吴茂贵 / 著

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

内 容 简 介

如果你是一位在校大学生，对大数据感兴趣，也知道使用的企业越来越多，市场需求更是日新月异，但苦于自己基础不够，心有余而力不足；也看过不少大数据方面的书籍、博客、视频等，但感觉进步不大；如果你是一位在职人员，但目前主要使用传统技术，虽然对大数据很有兴趣，也深知其对未来的影响，但因时间不够，虽有一定的基础，常常也是打两天鱼、晒三天网，进展不是很理想。

如果你有上述疑惑或遇到相似问题，本书正好比较适合你。本书从 OpenStack 云平台搭建、软件部署、需求开发实现到结果展示，以纵向角度讲解了生产性大数据项目上线的整个流程；以完成一个实际项目需求贯穿各章节，讲述了 Hadoop 生态圈中互联网爬虫技术、Sqoop、Hive、HBase 组件协同工作流程，并展示了 Spark 计算框架、R 制图软件和 SparkRHive 组件的使用方法。本书的一大特色是提供了实际操作环境，用户可以在线登录云平台来动手操作书中的数据和代码，登录网址请参考 http://www.feiguyun.com/support。

图书在版编目（CIP）数据

自己动手做大数据系统 / 张魁等著. —北京：电子工业出版社，2016.10
ISBN 978-7-121-29586-7

Ⅰ. ①自… Ⅱ. ①张… Ⅲ. ①数据处理系统 Ⅳ. ①TP274

中国版本图书馆 CIP 数据核字（2016）第 205183 号

策划编辑：符隆美
责任编辑：葛　娜
印　　刷：北京京师印务有限公司
装　　订：北京京师印务有限公司
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　　邮编：100036
开　　本：787×980　1/16　　印张：15.5　　字数：348 千字
版　　次：2016 年 10 月第 1 版
印　　次：2016 年 10 月第 1 次印刷
定　　价：49.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888
质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。
本书咨询联系方式：010-51260888-819　faq@phei.com.cn。

前　言

一个游泳爱好者，最大的烦恼是什么？没有好的教练？缺少好的教材？也许不是。如果哪天自己能拥有一个游泳池，可随时畅游，而且维护成本很低廉，甚至免费，同时还有教练的指导和一些游泳爱好者一起，那应该是一件很美的事。对于一个大数据爱好者，如果也能拥有一个属于自己的大数据实践环境，能够方便、快捷、随时随地使用真实环境，同时还有一些实战性、生产性的项目或课件，与一些志同道合的小伙伴一起攻坚克难，应该也是一件令人期待的事。

“纸上得来终觉浅，绝知此事要躬行”。要掌握一门技术，尤其像大数据相关技术，涉及的内容多，范围广，对环境的要求高，如果只是看看书、看看视频，很难深入理解，更不用说融会贯通了。一些有条件的学生，他们可以搭几个节点，组成一个微型大数据群，照着书中的一些实例练习，但这些练习往往支离破碎，缺乏系统性、生产性，更不用说包含生产性项目中的版本控制、质量管理和流程规范等。而这些对实施生产项目来说很重要，有时其重要性超过了对技术的要求。本书，就是为弥补这些内容而写的。

除了实战性、生产性的课件外，我们还提供了随时随地可操作、可实践的大数据云平台——飞谷云，这是我们自主开发的大数据平台，该平台用户可通过外网登录，与论坛及门户实现无缝连接。此外，还有很多志同道合的大数据爱好者一起学习、一起做项目。

本书主要内容

第 1 章，介绍我们为什么需要自己动手做大数据系统。

第 2 章，介绍动手做大数据系统的项目背景、项目架构及相关基础知识。

第 3 章，介绍大数据系统环境的搭建和配置，主要包括如何搭建和配置 Hadoop 集群、Sqoop、Hive、HBase、ZooKeeper、Spark、MySQL 等，图文并茂，内容翔实。

第 4 章，介绍大数据系统中数据获取相关技术，包括如何利用爬虫技术获取平面数据和使用

Sqoop 获取结构化数据。

第 5 章，介绍大数据系统中数据仓库工具 Hive 的使用方法及进行 ETL 的过程详解。

第 6 章，介绍大数据系统中数据库 HBase 的使用方法及和 Hive 之间的数据对接。

第 7 章，介绍如何使用数据展示利器 R 来展示 HDFS 中的数据。

第 8 章，介绍使用 Spark 计算模型来实时处理数据及 SparkRHive 组件的使用。

第 9 章，介绍如何搭建支撑大数据系统的云平台，以保证大数据系统的稳定性。

读者范围

- 对大数据感兴趣的院校师生。
- 对大数据有一定的基础，还想进一步熟悉整个生态系统的大数据爱好者。

勘误与支持

尽管我们仔细对待本书的写作，由于水平和能力有限，错误还是不可避免的。如果你在书中发现不妥或错误之处，请访问 http://www.feiguyun.com/support，留下宝贵意见，我们将非常感谢你的支持和帮助。

致谢

首先要感谢大数据实战团队，参与飞谷云大数据公益项目（www.feiguyun.com）的所有大数据爱好者，正是有了大家的支持和积极参与，才使得从飞谷一期的四个人，发展到目前飞谷七期的近四百人，短短一年多的时间，让我们真正感受到了共同坚持、诚信进取、协同分享的飞谷价值观所带来的收获和快乐，每期的项目线下启动会、交流会、项目结束总结会总能感受到大家积极参与的热情！同时也要感谢苏州大学计算机科学与技术学院何书萍老师、上海理工大学管理学院张帆老师、上海交通大学大数据分析俱乐部蒋军杰同学、中国社科院研究生院孙思栋同学、上海华师大数据分析俱乐部罗玉雪同学、上海大学黄文成同学等。

此外，要感谢飞谷管理团队的各位老师：陈健、刘军、吴嘉瑜、张勤池、王继红、张海峰、许小平、陶方震和刘李涛。诸君对飞谷大数据项目的热心参与及全力配合，是此公益项目得以持续推进的不懈动力。特别感谢为飞谷云提供实战项目的企业数据负责人；飞谷七期电商比价项目提供者——张晓雷先生及飞谷八期汽车推荐模型需求提供者——章水鑫先生，正是有了你们提供

的需求、数据和业务指导，才使得飞谷大数据小伙伴们有了学习大数据的真实场景，在实践中体会大数据分析价值和魅力。

飞谷云在全国一些大学还建立了交流群，作为每个群的组织者：中国科技大学张海洋同学、河南工程学院孟祥杰同学、南京农业大学邬家栋同学、西安电子科技大学刘东航同学等，为飞谷公益项目在院校中的推广，亦发挥了积极作用，在此一并表示感谢。

目　录

第 1 章
为什么要自己动手做大数据系统

1.1 大数据时代

“双 11”购物的狂欢、滴滴的便捷出行、阿尔法狗（AlphaGo）的进步、聊 QQ、刷微信、旅游、看病、健身等，我们已不由自主地置身于大数据时代，我们不但是大数据的创造者，同时也是大数据的体验者、受益者。随着 4G 的普及、移动互联网及“互联网+”的不断发展、5G 的临近、物联网的不断发展，大数据与我们的关系也越来越密切，取得的成果越来越鼓舞人心，对我们的影响深远。

2015 年 11 月初我国发布的《中共中央关于制定国民经济和社会发展第十三个五年规划的建议》提出，拓展网络经济空间，推进数据资源开放共享，实施国家大数据战略，提前布局下一代互联网，可以说这是我国推行的国家大数据战略。各招聘网的大数据人才需求日新月异，需要的大数据人数在不断增加，涉及的行业范围在不断扩展，开出的薪资也很诱人。总之，不管从国家层面还是企业、个人，大数据已经跟我们息息相关。

大数据的需求增长迅猛，但是大数据方面的人才供给却相对滞后。其原因有二：一是人才培养不够。据我们了解，现在开设大数据相关课程的院校屈指可数，有好消息是复旦大学于 2015 年 8 月成立了大数据学院，2016 年 9 月开始招生；二是目前大数据技术的应用主要集中在“BAT”类型的互联网公司，整个技术团队的维护成本很高，这也抬高了其他科技公司使用大数据技术的门槛。

现在很多大学生或在职人员学习大数据技术，大都通过网络、博客、购买书籍等方式学习，这种学习往往效率不高，虽然书中或网上有这些内容的介绍，但很多都是理论性或概要性的，缺少具体的、详细的内容或步骤。大数据相关技术要求的面较广，包括操作系统、开发语言、数据库、网络，甚至需要一定的英语基础，虽然大数据相关技术都是开源的，如 Hadoop、Sqoop、HBase、Redis、MongoDB、GraphDB、Hive、Spark、Storm、OpenStack 等，但即使是工作多年且有一定经验的在职人员，自己搭建大数据环境挑战也不少，其中不少因问题多半途而废，就更不用说在

校大学生了。

大数据技术实践性很强，仅仅看一些书或视频，没有实际操作，很难有深刻的理解，更不用说融会贯通了。缺乏实际操作，在具体实施项目时一遇到问题，往往会不知所措，不知如何分析、定位、处理问题，感叹书到用时方恨少。针对诸如此类的问题，我们希望通过本书对一个实战性项目的详细介绍，给正在或将学习大数据的同学或在职人员提供一些思路或帮助。

1.2 实战大数据项目

目前很多想学习或正在学习大数据的人，大都面临一些问题或困惑，本书的第一个特点就是系统性，覆盖了如何利用爬虫、Sqoop 等获取各种数据，如何利用 HDFS、HBase 等存储大数据，如何利用 MapReduce、Hive、Pig、Python、Spark 等技术来处理大数据，如何利用 Spark 及 R 分析展示大数据整个过程，而且这些过程我们都可以以实战项目的方式在云平台上完成，这又体现出本书的第二个特点，即操作的便捷性。同时，本书既有对大数据主要原理的概述，又不乏对一些关键细节的详细描述，如 SSH、NTP 等详细配置等，重点、难点部分还贴有源码，这体现了本书的第三个特点，即实战性。因此，这些内容降低了在校大学生学习、搭建、实践大数据的难度，对有一定工作经验的在职人员也有参考价值。

此外，关于大数据生态系统的每个组成部分，比如 Hadoop、Hive、HBase 等，专门论述的书籍有很多，本书没有对某个部分的使用方法做大而全的罗列，而是着重于组件之间如何配合工作，并以同一个案例贯穿其中，旨在让读者从宏观上对大数据实现技术有一个全面的了解。

大数据技术实践性很强，相关技术也都是开源免费的，其中包含很多组件，各个组件更新很快，相互间的关系也较紧密，大数据开发涉及知识面广、数据复杂度高。纸上得来终觉浅，如果仅仅看看书、视频、博客等，则只能了解一些表面现象，很难做到融会贯通，更不用说深刻理解了；绝知此事要躬行，要想对大数据技术有一个较全面和深刻的掌握，亲自实践必不可少，通过自己亲手搭建、开发和实践，将大大加深对各组件功能、原理、架构、关系等的理解，为系统后续维护、升级、二次开发、问题定位、问题排查、问题解决等打下良好的基础。

1.3 大数据演练平台

我记得 10 年前大学毕业找工作时，发现企业需要的各项计算机实操技术自己都没实际做过，还是依靠自己在实验室的 SQL 实践进入一家外企从事 DBA。进入企业后，确实感觉到企业的实际需求环境与大学学习环境有很多不同。后来依靠自己买书来学习并和互联网论坛上共同的技术交流群配合，实践了数据库、大数据开发，先后担任了某大型咨询公司的大数据开发经理，以及某互联网金融公司的大数据架构师的职务。

反过来思考，如果我们在刚毕业二、三年或在校时，就能接触到真正的“准生产”项目实践，有二、三年的实际工作经验，那么接下来的机会选择会好很多。这样的经历感受，我身边几位工作十多年的技术朋友都有同感，于是便有了飞谷云大数据公益项目，我们几个大数据爱好者秉持对 10 年前的自己负责的态度，来对待目前像 10 年前现状的在校学生，以及刚毕业不久的同学们，让大家有机会来实践在学校、实验室、公司接触不到的大数据实践技术。

据笔者了解，因为学生在校一般两年多，所以研究生导师一般接的项目偏理论和咨询，不愿涉足具体实战开发的项目，担心学生毕业后难以维护。而实验室一般是基于一个问题点来的，很难有实际商业需求的连贯性，且一般公司，也只是一个岗位对应一种技术，很难有机会接触全貌。但飞谷云环境正好能解决这个问题。

通过飞谷云大数据公益项目来真正参与进来，自己可以动手做起来。在我们的公益项目中有实际的企业大数据负责人做需求文档，飞谷云的导师作为技术负责人来做需求分析设计，参与的同学来做各岗位的工程师（数据获取工程师、数据模型工程师、数据分析工程师、数据可视化工程师等），项目每期都有启动会说明，中间有交流分享会，项目结束后有总结，以对应大家的技术实践全过程，真正让自己动手与团队动手达到协同分享的目的。

同时，大数据依赖的各项基础开发，比如 shell、Python、MySQL、爬虫技术、机器学习等，都会以基础班的形式同步支持项目技术储备，真正让即使完全什么都不懂的同学，只要感兴趣想学，就可以跟着本书和飞谷云大数据公益项目动手做起来。所以不管你是在校学生、大公司某一岗位的开发人员，还是中小公司的开发者，都可以通过本书和飞谷项目来实现在个人环境中所无法企及的收获与快乐。

第 2 章
项目背景及准备

2.1 项目背景

急速增长的数据使传统的关系型数据库无所适从，如何存储大数据、如何处理大数据，如何挖掘大数据，这是大数据时代面临的一些挑战。毫无疑问，在大数据处理中，Hadoop、Spark 已成为当下的王者技术，经过开源社区无数贡献者的强大推动，Hadoop 以其显著的低成本、高可靠性、高性能等特性，成功征服了众多大数据处理需求的商业机构和科研团体，既有像 Google、Facebook、Yahoo、阿里、百度、腾讯、华为等这样的知名企业，也有数以万计的中小企业。

大数据是一个大趋势，Hadoop 又是处理大数据的大趋势，大数据实战项目就是为学习、传播 Hadoop 技术而设立的。

2.2 项目简介

项目数据的来源主要包括大数据网站、网络金融、网上招聘、网购等。我们通过爬虫技术把这些信息抓取到 HDFS 或 MySQL 中，其他数据源通过 Sqoop 导入 HDFS 或 MySQL 中。然后使用 MapReduce、Spark 等技术对数据进行处理，处理后的数据导入 Hive、HBase 等，最后用 Java、HiveQL、R 及 Spark 等进行数据分析与展示，其间包括实施生产性项目的主要流程，如设计、开发、调试、问题定位及处理、版本管理等。一键实现环境的搭建，系统调度管理实现自动化，同时对系统的性能等实现实时监控。

2.3 项目架构

项目使用的主要技术是开源的，如 Hadoop、Hive、HBase、Spark、Ganglia、R、OpenStack、MySQL 等，系统的安装调度等用 Java、shell、Python 等工具自主开发。项目的总体架构如图 2.1 所示。

图 2.1　项目的总体架构图

2.4　操作系统

有关 Linux 和 UNIX 的知识网上有很多，这里主要介绍一些简单、实用的内容及本项目使用的操作系统，使初学者有一个大体的认识和了解。已熟悉 Linux 或 UNIX 的读者可跳过本节。

Linux 与 UNIX 的主要区别是：Linux 是开源、免费的，UNIX 是商业软件；Linux 能够运行在多种平台上，而 UNIX 大多与硬件配套。

常用的 Linux 及 Unix 版本信息如表 2.1 所示。

表 2.1　操作系统版本说明

类别	发行版本	说明	安装
Linux	Debain	一种流行的非商业性质的版本	apt-get install wget
	Ubuntu	Debain 的精炼版本 帮助文档 URL：help.ubuntu.com	apt-get install wget
	RedHat Enterprise	Red Hat Linux 商业化版本 帮助文档 URL：redhat.com/docs	yum install wget
	CentOS	模仿 Red Hat Enterprise Linux 的免费版本	yum install wget
	Fedora	从 Red Hat Linux 分出的非商业化版本	yum install wget
UNIX	Solaris	始于 Sun，但目前属于 Cracle 公司	安装软件包
	HP-UX	用于 HP 的硬件平台	安装软件包
	AIX	用于 IBM 的硬件平台 帮助文档 URL：www.redbooks.ibm.com	安装软件包

编写脚本的语言有 shell、Perl、Python 等。在大多数系统上，默认的 shell 都是 bash（即 Bourne Again shell），但在几种 UNIX 上也用 sh（Bourne shell）和 ksh（Korn shell）。各种操作系统上都

有 shell，用 shell 编写的脚步可移植性好，除了其调用的命令外，需要依赖的东西不多，但如果要实现一些较复杂或高端的脚本，建议采用 Perl 或 Python 等。

分析和理解 Linux 或 UNIX 系统中的一些常见日志文件，对于调试和排查问题大有帮助。如表 2.1 所示，列出了几种主要日志文件的用途，Linux 系统日志文件一般在/var/log 目录下，HP-UX 及 AIX 的一般在/var/adm 目录下。

表 2.2　日志文件列表

日志文件	涉及程序	操作系统	说明
mail*	与 mail 有关	所有系统	记录有关 mail 的信息
messages	很多	UR	系统日志文件
syslog*	很多	UH	系统日志主文件
cron	cron	RAH	记录 cron 的执行情况及出错信息
debug	很多	U	调试输出日志文件
daemon.log	很多	U	记录所有守护进程的功能消息
dpkg.log	dpkg	U	软件包管理日志
/etc/httpd*	httpd	R	记录 Apache HTTP 服务器日志
wtmp	login	所有系统	用户登录记录

系统说明：
U=Ubuntu，R=Red Hat 和 CentOS，H=HP-UX，A=AIX

本项目使用的操作系统为 Ubuntu 14.0.4。

登录 Ubuntu 服务器，可通过 GNU 工具 PuTTY 客户端，配置如图 2.2 所示。

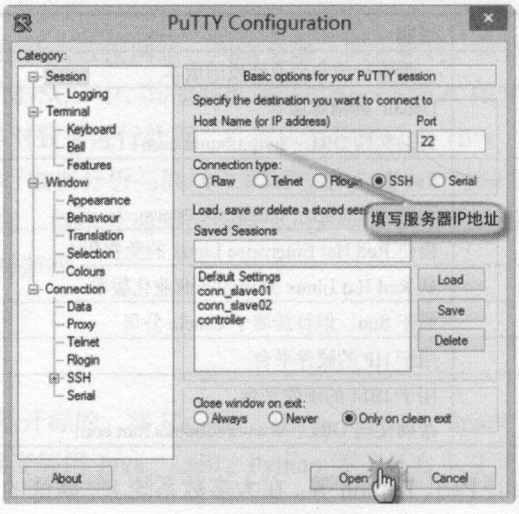


图 2.2　PuTTY 配置

单击 Open 按钮，进入 Linux 界面，如图 2.3 所示。

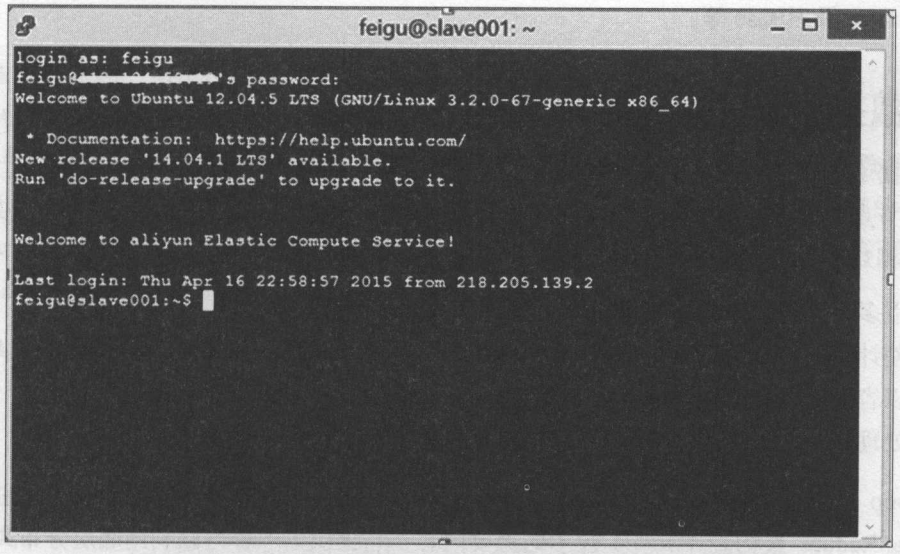


图 2.3　Linux 界面

2.5　数据存储

目前国内大数据常用“4V”来定义，即大容量（Volume）、多类型（Variety）、高时效（Velocity）、价值密度低（Value）。大数据增长对数据存储带来的挑战不仅体现在存储容量方面，还涉及数据管理、数据处理方面，迎接这些挑战带来了数据处理技术的革命，并由此开启了大数据时代。

为了克服大数据对数据存储带来的挑战，人们对传统存储方式进行了不少革新，在提升存储容量、系统吞吐量、数据可靠性、数据维护等方面取得了较大的进展。

如表 2.3 所示是传统 DBMS 与大数据存储方面的一些比较。

表 2.3　数据存储模式新旧对比

对比项	传统模式	新模式
存储方式	行存储	列存储、行存储
可扩展性	基于分区、分布式缓存	基于 HDFS，线性扩展
可靠性	热备、归档等	内建的复制机制
成熟度	高	发展更新阶段
处理特性	业务数据的实时处理	大数据实时批量处理
设计	主键、联合、元数据与应用数据在一起、SQL	行键、弱规范化、元数据与应用数据分离、NoSQL

本项目主要以 HDFS 格式存放数据，部分数据存入 MySQL 数据库，然后对这些数据进行处理后存入 Hive 及 HBase 等。

2.6 数据处理

大数据不仅体现在数量上，也体现在复杂度上，为了应对大数据的挑战，主要从两方面入手：增加计算节点的处理能力和优化数据处理的架构。目前大数据处理采用分布式、并行处理的架构，这种架构相对于传统数据处理技术有如下创新：

- 以多节点协同代替单节点能力的提升；
- 使计算与数据的结合更紧密和科学；
- 以容错机制代替对低故障率的要求；
- 处理架构的平滑扩展。

Hadoop 的大数据处理最初采用 MapReduce，从 2012 年开始进入 YARN，目前正进入 Spark 时代。MapReduce 架构是 Hadoop 技术的重要内容，它提供了一种可以利用底层分布式处理环境进行并行处理的模式，并将处理过程分为两个阶段：Map 和 Reduce。在 Map 阶段，原始数据通过分片处理输入到 Mapper 进行过滤和转换，生成的中间数据在 Reduce 阶段作为 Reducer 的输入，经过 Reducer 的聚合处理，得到输出结果。MapReduce 数据处理架构如图 2.4 所示。

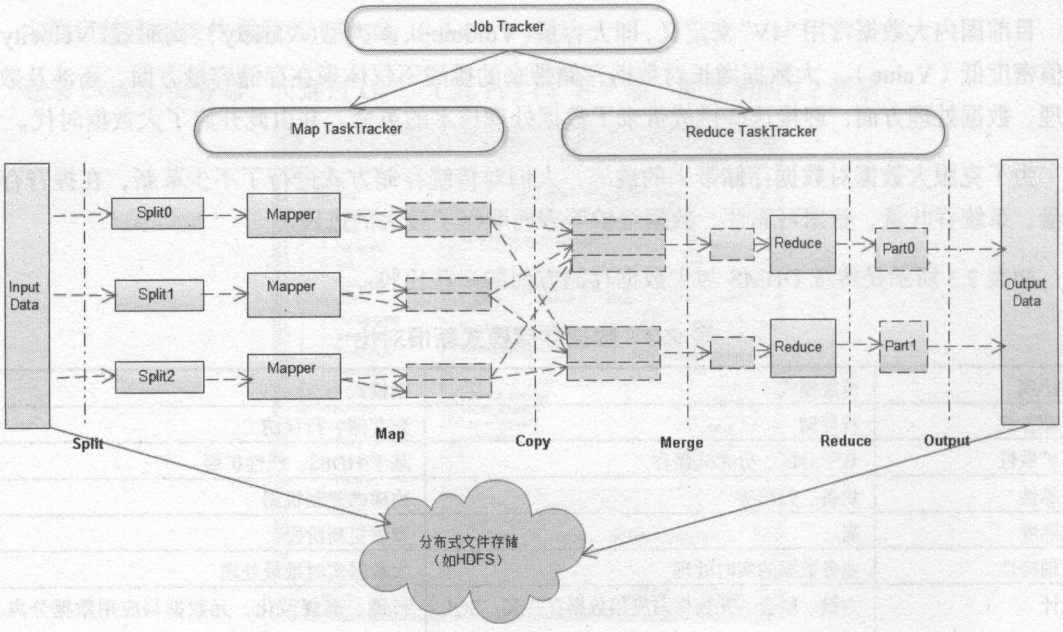


图 2.4 MapReduce 数据处理架构图

YARN 最核心的改进是把 MapReduce 架构中的资源管理和作业调度两个功能拆分到独立的进程中。

Spark 基于 Map、Reduce 算法实现的分布式计算，拥有 Hadoop MapReduce 所具有的优点；但不同于 MapReduce 的是，Job 中间输出结果可以保存在内存中，从而不再需要读写 HDFS，因此 Spark 能更好地适用于数据挖掘与机器学习等需要迭代的 Mapreduce 的算法。其数据源可以是 HDFS、HBase 等。其实也可以说 MapReduce、YARN、Spark 三者代表了 Hadoop 发展的几个阶段，目前 Spark 非常火，是用 Scala 语言写的。从 Spark 官方文档获知，在内存运行方面 Spark 比 MapReduce 快近 100 倍，基于硬盘运行是 MapReduce 的 10 倍。

Spark 的架构如图 2.5 所示。

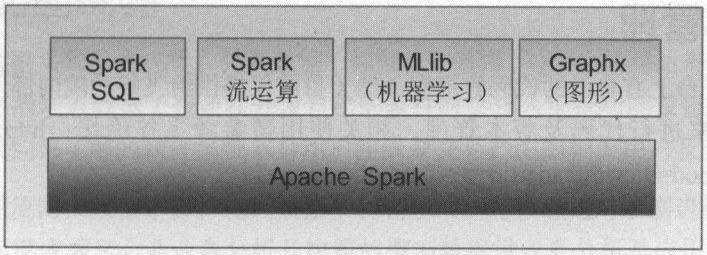


图 2.5　Spark 架构图

本项目数据处理主要采用 YARN 及 Spark 等技术。

2.7　开发工具

Java 安装版本说明如下：

Hadoop 生态系统中的诸多应用，都是用 Java 语言开发的。本项目安装环境中使用的是 JDK 1.8 版本。安装后的目录位置是/home/hadoop/bigdata/java/。此外，本书中所涉及的 Java 开发代码也基于此版本。

如果读者机器上已经安装了 JDK，但是不确定是什么版本，可以使用“java -version”命令来查看版本信息。

Python 安装版本说明如下：

在本书 4.1 节中，会讲到使用 Python 语言实现网络爬虫的功能，所使用的 Python 版本是 2.7.3 开发版。飞谷云使用的操作系统是 Ubuntu 和 CentOS，其默认安装的 Python 版本是 2.7.3。

在第 4 章开始部分，我们要安装 Scrapy 运行环境，会重新安装 Python。读者可以使用

“python -V”或“python --version”命令来查看当前环境使用的 Python 版本。

2.8 调试工具

Java 模块代码的开发，使用了 Eclipse 作为调试工具。读者也可以使用其他开发工具来进行调试。

Python 部分爬虫功能的开发，使用了 Eclipse 下的 Python 插件。关于 Python 的 IDE 工具有很多，有些还提供断点调试/单步跟踪功能。但是本项目中涉及的爬虫代码，是在 Scrapy 框架内实现的。Scrapy 在执行时采用多线程方式，建议读者多使用日志输出来进行调试。

2.9 版本管理

在飞谷云平台上有自己的代码管理服务器（目前仅提供给飞谷学院内部使用，未开放）gitlab 社区版服务器来进行代码及版本管理，相关使用请参考飞谷论坛：http://www.feiguyun.com/bbs/forum.php?mod=viewthread&tid=8940&extra=page=1。

第 3 章
大数据环境搭建和配置

3.1 各组件功能说明

大数据系统的数据采集分为三类：公共数据（如微信、微博、QQ 公共网站等公开的互联网数据）、企业应用程序的埋点数据（企业在开发自己的软件时会接入记录功能按钮及页面的点击等行为数据），以及我们都了解的软件系统本身用户注册及交易产生的相关用户及交易数据。

这三类数据共同流向大数据平台，在大数据平台利用维度建模理论对数据进行 ETL（抽取、转换、加载），从而形成满足企业业务分析及决策的模型数据供业务分析及应用系统调用。具体的大数据系统的数据流程图及组件如图 3.1 所示。

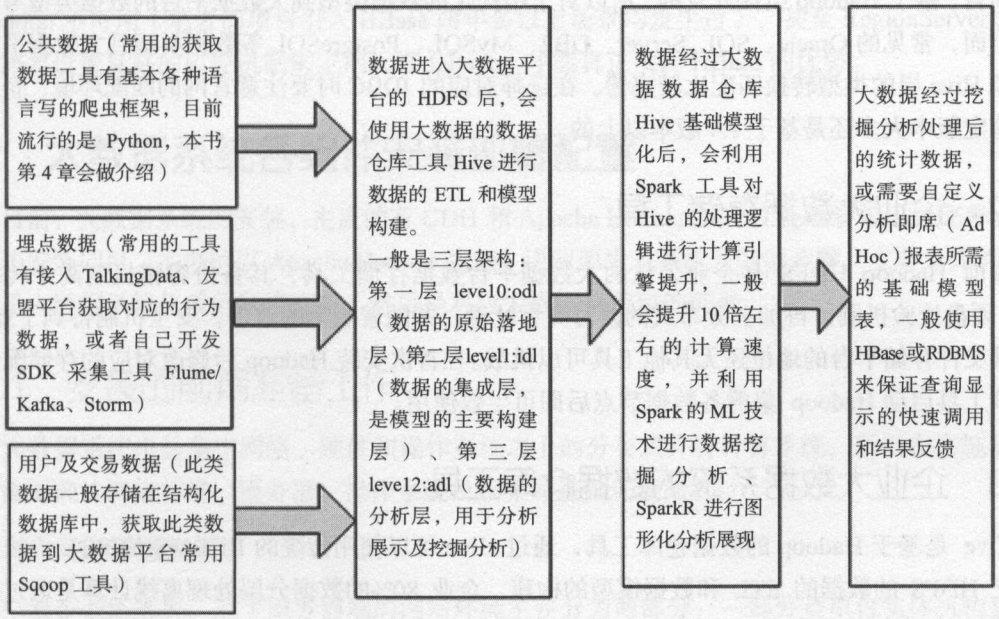


图 3.1　大数据系统的数据流图及组件

通过图 3.1 可以看出大数据系统主要有以下几种工具。

3.1.1 各种数据源的采集工具

1. 公共数据采集工具

公共数据采集工具 Python 是比较经典的脚本语言，在大数据系统开发中常用作公共数据采集的爬虫工具、自动化部署安装、运维的开发工具，当然也可以用来进行数据分析开发。

2. 埋点数据采集工具

企业日志信息采集工具一般有 Flume、Kafka 和 Storm。目前 Flume 是互联网公司使用的比较多且相对稳定、易用的数据采集工具，该工具的优势是配置简单（直接定义数据源、数据通道和数据落地位置就可以了），并支持数据处理插件开发。而 Kafka 是一个流式数据采集工具，该工具的采集频率会明显高于 Flume，一般企业里对 Flume 的采集频率都在分钟级别配置，而 Kafka 却是近实时的大数据采集，主要是采集数据到队列，有可能会出现重复数据，可以结合 Storm 的事务机制来进行数据处理，保证数据不丢失。

3. 企业结构化数据采集工具

目前 Sqoop 是企业使用的比较成熟的同步结构化数据到大数据平台的工具，是 Apache 的一个顶级项目，基于 Hadoop 组件开发的，所以对于结构化的数据类型到大数据平台的数据类型考虑得比较全面，常见的 Oracle、SQL Server、DB2、MySQL、PostgreSQL 等数据类型到大数据平台数据仓库 Hive 层的类型转换匹配比较完善。在选择对应的 JDBC 时要注意官网的匹配环境，企业使用的稳定版本大多还是基于 1.4 版本以上的。

3.1.2 企业大数据存储工具

目前 Hadoop HDFS 是企业公认的大数据平台数据存储工具，其备份容错机制高于传统的 RAID 硬盘校验机制，再加上分布式的读写高吞吐量，并随着版本的改进，安全机制得到了加强，目前其文件存储平台的地位还无其他工具可以挑战。在首次安装 Hadoop 后修改对应的存储配置文件，该工具启动 Hadoop 集群各数据节点后即可生效使用。

3.1.3 企业大数据系统的数据仓库工具

Hive 是基于 Hadoop 的数据仓库工具，通过 Hive 可以使用传统的 RDBMS 的 SQL 语法来实现基于 HDFS 的数据的 ETL 和数据模型的构建。企业 80%的数据分层处理离线计算是通过基于 Hive 的 SQL 来构建的，并且 Hive 也支持 Spark 的计算引擎接口和分析展示的 R 包接口（RHive）

来获取 Hive 构建好的模型表及逻辑。

3.1.4　企业大数据系统的分析计算工具

目前 Spark 是公认的会取代 Hadoop 的 MapReduce 框架的大数据分析计算工具，其基于 Scala 函数式编程的语言和 RDD（弹性分布式数据集）的架构设计，使其 Spark SQL 计算速度是 Hive 的 10 倍以上，加上其 Spark 机器学习组件，SparkR 组件更是可以基于大数据存储系统直接进行计算及分析产出，大大提高了大数据系统处理海量数据的效率。不足是 Spark 在具体的产线应用中还有不少需要调节修复的 Bug 等，相对 Hive 来说稳定性还是不够强健。但其版本更新和 Bug 修复的速度也很快，所以备受互联网公司的青睐。

3.1.5　企业大数据系统的数据库工具

如果说 Hive 是大数据系统中基于 OLAP（On-Line Analysis Processing，联机分析处理）的数据仓库工具，那么可以把 HBase 比作基于 OLTP（On-Line Transaction Processing，联机事务处理）的工具，其基于列式的存储机制可以方便地从上亿条数据中快速查询到对应的 rowkey 值，并且 TPS 可以达到万级别以上，这是普通的 RDBMS 系统所不能突破的瓶颈，并且 HBase 可以通过参数的设置做到满足业务时间内的数据一致性，从而满足基于实时风控、实时精准营销等场景的在线系统应用。但由于 HBase 的文件合并机制往往会影响到服务应用，在 GC 和数据恢复中也会导致 Region 不可用（笔者的项目引入 HBase 两年多以上场景均发生过），甚至 RegionServer 挂掉，因此需要设置良好的监控机制来定期运维 HBase，才能真正发挥 HBase 的优势。

3.2　大数据系统各组件安装部署配置

目前，大数据系统的安装，主流的有 CDH 和 Apache Hadoop。CDH 各兼容包已经完善，只需要按步安装即可。本书就以 Apache Hadoop 为例，说明原生态系统安装的步骤，主要分为两部分，首先是大数据系统依赖的环境变量，然后是对应大数据安装包的部署。

3.2.1　安装的前期准备工作

大数据系统也是基于网络、硬件和操作系统之上的分布式并行计算系统，所以在实施安装部署前需要确认网络环境、服务器、操作系统及相关的基础环境配置。

1．网络环境确认

一般企业级部署，对于服务器端的网络环境主要分为两部分：一部分是单台实体机构建的安全网络环境；另一部分是由一批实体机构建的云网络环境。对于大数据系统的部署，会根据生产

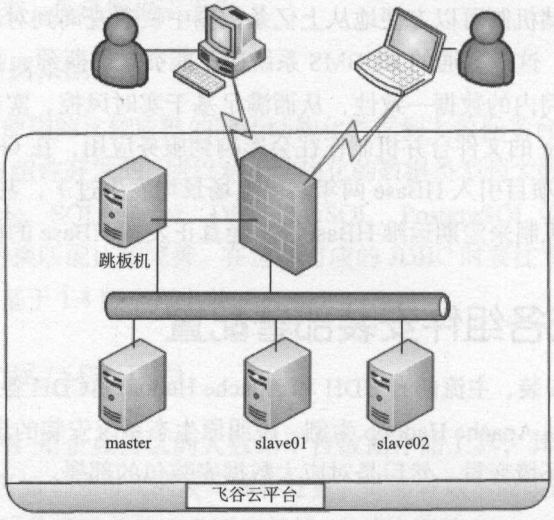
业务实际情况来选择部署不同的网络环境。一般由于实时的大数据系统需要支持在线应用的精准营销系统、推荐系统、风控系统对应的实时接口调用，所以会选择在与在线系统同万兆网卡网段的实体机上部署大数据系统。而非实时的大数据系统可以基于一批实体机构建的云网络上的虚机服务器来部署，可以充分利用云计算的资源池化按需分配各大数据集群节点的配置，并且可以利用网络分发节点必需的基础环境软件。

本书中的服务器端网络环境就是后者，在 IDC 机房的批量服务器上构建的云平台网络环境。具体细节请参考第 9 章“自己动手搭建支撑大数据系统的云平台”。

对于连接部署环境的客户端，通常与生产环境服务器不在一个网络中，而是通过跳板机的方式连接到生产或实战环境网络。飞谷云的客户端连接也是用户基于公网客户端连接到飞谷云的跳板机，再进入到对应的生产及实战环境的。

2. 服务器确认

在部署大数据系统前需要规划好对应的集群节点及相应功能，本书实例规划如图 3.2 所示。



Host-IP	HostName	CPU	MEM	HDFS	YARN	对应大数据组件	备注
192.168.0.166	master	2	8GB	NameNode	ResourcesManager	HBase Master	集群主节点
192.168.0.167	slave01	2	8GB	DataNode	NodeManager	Hive、Spark、HBase Region、Sqoop	计算调度
192.168.0.168	slave02	2	8GB	DataNode	NodeManager	Hive、Spark、HBase Region	数据计算节点

图 3.2　实例规划

以上客户端通过跳板机进入飞谷云环境。

3. 操作系统确认

使用“uname -a”确认操作系统相关参数：

```
[hadoop@master /]$ uname -a
Linux master 2.6.32-504.30.3.el6.x86_64 #1 SMP Wed Jul 15 10:13:09 UTC
 2015 x86_64 x86_64 x86_64 GNU/Linux
```

可知，本书是基于 CentOS 64 位系统来搭建 Hadoop 环境的。

4. 基础环境配置确认

确认操作系统字符集是否符合业务需要：

```
[hadoop@master ~]$ echo $LANG
en_US.UTF-8
[hadoop@master ~]$
```

确认在操作系统下计算机名称是否已经更改、安装用户是否已经创建，如果没有创建安装用户，则需要创建用于安装 Hadoop 的用户如 hadoop，检查对应的 hadoop 用户的磁盘读写权限、磁盘空间，确保该用户可以安装部署 Hadoop 系统。

（1）创建用户组（创建用户组 hadoop）：

```
groupadd hadoop
```

（2）创建用户（创建用户 hadoop，并将用户添加到用户组中）：

```
useradd -m -g hadoop hadoop
echo -e " hadoop \n hadoop " | passwd hadoop
```

（3）修改计算机名称（更改后保存退出）：

```
vi /etc/hostname
```

```
[hadoop@master /]$ vi /etc/hostname
[hadoop@master /]$ cat /etc/hostname
master
```

创建完毕，可以用 hadoop 用户登录验证用户及权限：

```
[hadoop@master ~]$ whoami
hadoop
[hadoop@master ~]$ umask
0002
```

3.2.2　Hadoop 基础环境安装及配置

网络、硬件和操作系统层面确定完毕后，就可以下载 Hadoop 软件包进行安装部署了，并且在

系统网络之间需要 NTP（网络时间协议）保持时间同步。

1. 安装文件准备

对应的安装介质如下。

jdk-8u60-linux-x64.rpm，官方下载地址：

http://www.oracle.com/technetwork/java/javase/downloads/index-jsp-138363.html

hadoop-2.7.1.tar.gz，官方下载地址：

http://www.apache.org/dyn/closer.cgi/hadoop/common/hadoop-2.7.1/hadoop-2.7.1.tar.gz

2. 配置环境变量及无密码传输

（1）在 Master 节点进行如下设置：

```
vi /etc/hosts
192.168.0.166 master
192.168.0.167 slave01
192.168.0.168 slave02
```

验证：

```
cat /etc/hosts
```

```
[hadoop@master /]$ cat /etc/hosts
127.0.0.1   localhost localhost.loc
ldomain4
::1         localhost localhost.loc
ldomain6
192.168.0.166 master
192.168.0.167 slave01
192.168.0.168 slave02
```

（2）对用户环境变量进行如下设置：

```
env
```

配置对应的用户环境变量：

```
vi /home/hadoop/.bashrc
export JAVA_HOME=/usr/java/jdk1.8.0_60
export HADOOP_HOME=/home/hadoop/bigdata/hadoop
export HADOOP_USER_NAME=hadoop
export PATH=$JAVA_HOME/bin:$HADOOP_HOME/bin:$HADOOP_HOME/sbin:$PATH
```

验证：

```
[hadoop@master /]$ env
HOSTNAME=master
SELINUX_ROLE_REQUESTED=
TERM=xterm
SHELL=/bin/bash
HADOOP_HOME=/home/hadoop/bigdata/hadoop
HISTSIZE=1000
SSH_CLIENT=192.168.0.172 54862 22
SELINUX_USE_CURRENT_RANGE=
ZK_HOME=/home/hadoop/bigdata/zk
SSH_TTY=/dev/pts/0
USER=hadoop
```

在以上设置中，JAVA_HOME 表示 Java 安装目录，HADOOP_HOME 表示 Hadoop 安装目录，HADOOP_USER_NAME 表示安装 Hadoop 时需要用到的用户，PATH 表示 Hadoop 需要引用的启动路径包添加进系统启动程序内。

（3）在 master 节点生成公钥，放到其余各节点上，实现 master 到各节点的无密码传输：

```
[hadoop@master .ssh]$cd /home/hadoop/.ssh
[hadoop@master .ssh]$ ssh-keygen
```

验证：

```
[hadoop@master .ssh]$cd /home/hadoop/.ssh
[hadoop@master .ssh]$ ls -la
```

```
[hadoop@master .ssh]$ ls -la
total 48
drwx------. 2 hadoop hadoop  4096 Jan 20 14:25 .
drwx------. 7 hadoop hadoop  4096 Jan 30 07:04 ..
-rw-r--r--. 1 hadoop hadoop   396 Jan 19 06:38 167.pub
-rw-r--r--. 1 hadoop hadoop   396 Jan 19 06:38 168.pub
-rw-------. 1 hadoop hadoop  5146 Jan 23 13:44 authorized_keys
-rw-------. 1 hadoop hadoop  1675 Jan 22 13:58 id_rsa
-rw-r--r--. 1 hadoop hadoop   395 Jan 22 13:58 id_rsa.pub
```

把 master 节点上的公钥文件 rsa.pub 内容添加进 authorized_keys，然后分发到各个子节点上（因为 master 需要无密码启动各个子节点上的 bin 程序，所以 master 是 SSH 客户端，各个子节点是 SSH 验证服务端）。

```
[hadoop@master .ssh]$cd /home/hadoop/.ssh
[hadoop@master .ssh]$cat rsa.pub >> authorized_keys
[hadoop@master .ssh]$scp authorized_keys hadoop@slave01:/home/hadoop/.ssh/
[hadoop@master .ssh]$scp authorized_keys hadoop@slave02:/home/hadoop/.ssh/
```

注意：.ssh 目录、公钥、私钥的权限都有严格的要求，需要确认各节点对应的权限是否正确。

- 用户目录的权限是 755 或 700，不能是 77*；

- .ssh 目录的权限是 755；
- .pub 或 authorized_key 的权限是 644；
- 私钥的权限是 600。

验证：

```
[hadoop@master .ssh]$ ssh slave01
Last login: Fri Mar 11 08:12:57 2016 from master
[hadoop@slave01 ~]$
```

3. 配置 Hadoop 的对应参数

将下载的 Hadoop 及 Java 文件上传到 master 节点上，运行以下命令解压缩安装：

```
[hadoop@master .ssh]$yum install jdk-8u60-linux-x64.rpm
tar -zxf hadoop-2.7.1.tar.gz
mkdir bigdata
mv hadoop-2.7.1 bigdata/
cd bigdata/
mv hadoop-2.7.1 hadoop
```

然后，可以更改 hadoop 目录下的配置文件参数。

提示：Hadoop 的配置文件基本上都是 XML 文件，所有的配置都放在<configuration>和</configuration>标签之间，一个<configuration>标签里面可以存在多个<property>和</property>标签。<name>标签里面就是我们想要设定的属性名称；<value>标签里面是我们想要设定的值；<description>标签是描述属性作用的，可以不写。绝大多数配置都是在 XML 文件里面设置的，因为在这里做的配置对全局用户都生效，而且是永久的。并且，Hadoop 的配置文件是最底层的（第一层），Hive、HBase、Spark 配置参数是第二层，基于会话的命令行设置是第三层，用户自定义配置（第三层）会覆盖默认配置（第二层或第一层）。

更改 core-site.xml 参数（对应的参数值）：

```
vi /home/hadoop/bigdata/hadoop/etc/hadoop/core-site.xml
<configuration>
<property>
<name>fs.defaultFS</name>
<value>hdfs://master:9000/</value>
</property>
<property>
        <!-- 设置每个节点上的临时文件目录 -->
        <name>hadoop.tmp.dir</name>
        <!-- 当前用户需要对此目录有读写权限，启动集群时自动创建  -->
        <value>/home/hadoop/bigdata/data/hadoop/tmp</value>
```

```
    </property>
</configuration>
```

更改 hdfs-site.xml 参数（HDFS 文件系统对应的参数值）：

```
vi /home/hadoop/bigdata/hadoop/etc/hadoop/hdfs-site.xml
<configuration>
<property>
    <!-- 指定 SecondaryNamenode 所在地址。本例设为和 NN 位于同一个主机 -->
    <name>dfs.namenode.secondary.http-address</name>
    <value>master:9001</value>
</property>
<property>
<name>dfs.datanode.data.dir</name>
<value>file:/home/hadoop/bigdata/data/hadoop/hdfs/datanode</value>
</property>
<property>
<name>dfs.namenode.name.dir</name>
<value>file:/home/hadoop/bigdata/data/hadoop/hdfs/namenode</value>
</property>

<property>
<name>dfs.replication</name>
<value>3</value>
</property>
</configuration>
```

更改 mapred-site.xml 参数：

```
vi /home/hadoop/bigdata/hadoop/etc/hadoop/mapred-site.xml
<configuration>
<property>
<name>mapreduce.framework.name</name>
<value>yarn</value>
</property>
</configuration>
```

更改 yarn-site.xml 参数：

```
vi /home/hadoop/bigdata/hadoop/etc/hadoop/yarn-site.xml
<configuration>

<!—更改 YARN 资源的配置属性信息-->
<property>
<name>yarn.resourcemanager.hostname</name>
<value>master</value>
</property>
<property>
<name>yarn.nodemanager.aux-services</name>
<value>mapreduce_shuffle</value>
```

```
</property>
</configuration>
```

更改 slaves 文件参数：

```
vi /home/hadoop/bigdata/hadoop/etc/hadoop/slaves
slave01
slave02
```

以上配置信息更改完毕后，把对应的.bashrc、/etc/hosts 及/home/hadoop/bigdata/hadoop 目录都复制到 slave01、slave02 对应的目录下：

```
scp /home/hadoop/.bashrc hadoop@slave01:/home/hadoop/
scp /etc/hosts hadoop@slave01:/etc/hosts
scp -r /home/hadoop/bigdata/hadoop hadoop@slave01:/home/hadoop/ bigdata/
scp /home/hadoop/.bashrc hadoop@slave02:/home/hadoop/
scp /etc/hosts hadoop@slave02:/etc/hosts
scp -r /home/hadoop/bigdata/hadoop hadoop@slave02:/home/hadoop/ bigdata/
```

注：以上仅是 Hadoop 安装的基本参数配置，关于资源池划分及相关性能参数要根据具体业务进行相应的配置，具体参数的使用场景会在飞谷公益项目中进行划分予以深入解析。

4. 启动并验证 Hadoop 安装

启动对应的 Hadoop 程序：

```
[hadoop@master hadoop]$ cd /home/hadoop/bigdata/hadoop/sbin [hadoop@master sbin]$ sh start-all.sh
```

显示如下：

```
[hadoop@master sbin]$ sh start-all.sh
This script is Deprecated. Instead use start-dfs.sh and s
16/03/11 09:39:33 WARN util.NativeCodeLoader: Unable to l
doop library for your platform... using builtin-java clas
licable
Starting namenodes on [master]
master: starting namenode, logging to /home/hadoop/bigdat
/hadoop-hadoop-namenode-master.out
slave01: starting datanode, logging to /home/hadoop/bigda
s/hadoop-hadoop-datanode-slave01.out
slave02: starting datanode, logging to /home/hadoop/bigda
s/hadoop-hadoop-datanode-slave02.out
Starting secondary namenodes [master]
master: starting secondarynamenode, logging to /home/hado
doop/logs/hadoop-hadoop-secondarynamenode-master.out
16/03/11 09:39:57 WARN util.NativeCodeLoader: Unable to l
```

```
[hadoop@master sbin]$ jps
1590 SecondaryNameNode
2038 Jps
1751 ResourceManager
1399 NameNode
[hadoop@master sbin]$
```

```
[hadoop@master sbin]$ ssh slave01
Last login: Fri Mar 11 08:24:36 2016 from master
[hadoop@slave01 ~]$ jps
1490 Jps
1342 NodeManager
1231 DataNode
[hadoop@slave01 ~]$ ssh slave02
Last login: Fri Mar 11 08:12:05 2016 from 192.168.0.172
[hadoop@slave02 ~]$ jps
1314 NodeManager
1203 DataNode
1462 Jps
[hadoop@slave02 ~]$
```

```
[hadoop@slave02 ~]$ hdfs dfsadmin -report
16/03/11 09:45:13 WARN util.NativeCodeLoader: Unable
doop library for your platform... using builtin-java
licable
Configured Capacity: 19391348736 (18.06 GB)
Present Capacity: 5039931392 (4.69 GB)
DFS Remaining: 5036429312 (4.69 GB)
DFS Used: 3502080 (3.34 MB)
DFS Used%: 0.07%
Under replicated blocks: 74
Blocks with corrupt replicas: 0
Missing blocks: 0
Missing blocks (with replication factor 1): 0
```

3.2.3　Hive 安装及配置

1. 安装文件准备

下载 Hive 安装包：

http://apache.fayea.com/hive/hive-1.2.1/apache-hive-1.2.1-bin.tar.gz

根据 OS 版本下载对应的 MySQL 服务器和客户端。

2. 配置环境变量和安装 MySQL

根据 OS 版本使用下载命令直接下载对应的软件包：

MySQL-client-5.5.46-1.linux2.6.x86_64.rpm、MySQL-server-5.5.46-1.linux2.6.x86_64.rpm

在 Ubuntu 环境下，使用以下命令安装对应的 MySQL：

```
[hadoop@master hadoop]$ sudo apt-get install mysql-server mysql-client
```

在 CentOS 环境下，使用以下命令安装 MySQL：

```
[hadoop@master hadoop]$ rpm -ivh MySQL-client-5.5.46-1.linux2.6.x86_64.rpm
[hadoop@master sbin]$ rpm -ivhMySQL-server-5.5.46-1.linux2.6.x86_64.rpm
```

```
#在服务目录下启动 MySQL 服务
service mysql start
#创建用户密码并登录
/usr/bin/mysqladmin -uroot password '123456
mysql -uroot -p123456
#创建 Hive 元数据库并授权 hive 用户访问
create database hivemeta
alter database hivemeta character set latin1;
grant all privileges on *.* to hive@"%" identified by "123456" with grant option;
flush privileges;
```

启动 MySQL，创建 Hive 元数据库并做相关配置。

3. 配置 Hive 参数

配置 Hive 的系统环境变量：

```
vi /home/hadoop/.bashrc
export HIVE_HOME=/home/hadoop/bigdata/hive
export PATH=$HIVE_HOME/bin:$PATH
source /home/hadoop/.bashrc
scp .bashrc hadoop@192.168.0.167:/home/hadoop/
scp .bashrc hadoop@192.168.0.168:/home/hadoop/
```

验证（在各节点上确认默认路径）：

```
[hadoop@master ~]$ echo $HIVE_HOME
/home/hadoop/bigdata/hive
```

解压缩文件并更改配置参数文件：

```
cd /home/hadoop/bigdata/
tar -zxf apache-hive-1.2.1-bin.tar.gz
mv apache-hive-1.2.1-bin hive
cd /home/hadoop/bigdata/hive/conf
cp hive-default.xml.template hive-site.xml
cp hive-env.sh.template hive-env.sh
cp hive-log4j.properties.template hive-log4j.properties
```

各配置文件主要参数更改如下：

```
#设置 Hive 的环境配置目录及 hadoop 目录，vi 编辑输入后按 wq!保存退出
```

```
vi /home/hadoop/bigdata/hive/conf/hive-env.sh
export HADOOP_HOME=/home/hadoop/bigdata/hadoop
export HIVE_CONF_DIR=/home/hadoop/bigdata/hive/conf

#设置 Hive 的日志存储路径，vi 编辑输入后按 wq!保存退出
vi /home/hadoop/bigdata/hive/conf/hive-log4j.properties
hive.log.threshold=ALL
hive.root.logger=INFO,DRFA
hive.log.dir=/home/hadoop/bigdata/hive/log
hive.log.file=hive.log

#设置 Hive 运行文件路径及元数据库连接信息，vi 编辑输入后按 wq!保存退出
vi /home/hadoop/bigdata/hive/conf/hive-site.xml
<property>
  <name>hive.metastore.warehouse.dir</name>
  <value>hdfs://master:9000/hive/warehouse</value>
</property>
<property>
  <name>hive.exec.scratchdir</name>
  <value>hdfs://master:9000/hive/scratchdir</value>
</property>
<property>
  <name>hive.querylog.location</name>
  <value>/home/hadoop/bigdata/hive/logs</value>
</property>
<property>
  <name>javax.jdo.option.ConnectionURL</name>
  <value>jdbc:mysql://master:3306/hivemeta?createDatabaseIfNotExist=true</value>
</property>
<property>
  <name>javax.jdo.option.ConnectionDriverName</name>
  <value>com.mysql.jdbc.Driver</value>
</property>
<property>
  <name>javax.jdo.option.ConnectionUserName</name>
  <value>hive</value>
</property>
<property>
  <name>javax.jdo.option.ConnectionPassword</name>
  <value>123456</value>
</property>
<property>
  <name>hive.metastore.local</name>
  <value>false</value>
</property>
<property>
  <name>hive.metastore.uris</name>
  <value>thrift://master:9083</value>
</property>
```

4. 验证 Hive 相关服务

Hive 安装完成后需要启动对应的 metastore 服务和 Hiveserver2 服务，即可执行 Hive 的操作任务。

```
#在 metastore 所在服务器 master 启动 metastore 服务
[hadoop@master bin]$ nohup hive --service metastore&
#在 Hiveserver2 所在服务器 slave01 启动 Hiveserver2 服务
[hadoop@slave01 bin]$ nohup hive --service hiveserver2&
```

验证——查看相关服务，以及进入相关数据库和表：

```
[hadoop@master bin]$ nohup hive --service metastore&
[1] 32268
```

```
[hadoop@master bin]$ ps -ef |grep Hivemeta
hadoop   32432 32248  0 02:36 pts/4    00:00:00 grep Hi
[hadoop@master bin]$ ps -ef |grep Hive
hadoop   32268 32248  2 02:35 pts/4    00:00:15 /usr/ja
bin/java -Xmx256m -Djava.net.preferIPv4Stack=true -Dhac
me/hadoop/bigdata/hadoop/logs -Dhadoop.log.file=hadoop.
me.dir=/home/hadoop/bigdata/hadoop -Dhadoop.id.str=hadc
t.logger=INFO,console -Djava.library.path=/home/hadoop/
lib/native -Dhadoop.policy.file=hadoop-policy.xml -Djav
4Stack=true -Xmx512m -Dhadoop.security.logger=INFO,Null
ache.hadoop.util.RunJar /home/hadoop/bigdata/hive/lib/h
.1.jar org.apache.hadoop.hive.metastore.HiveMetaStore
```

```
[hadoop@slave01 bin]$ nohup hive --service hiveserver2&
[1] 5054
```

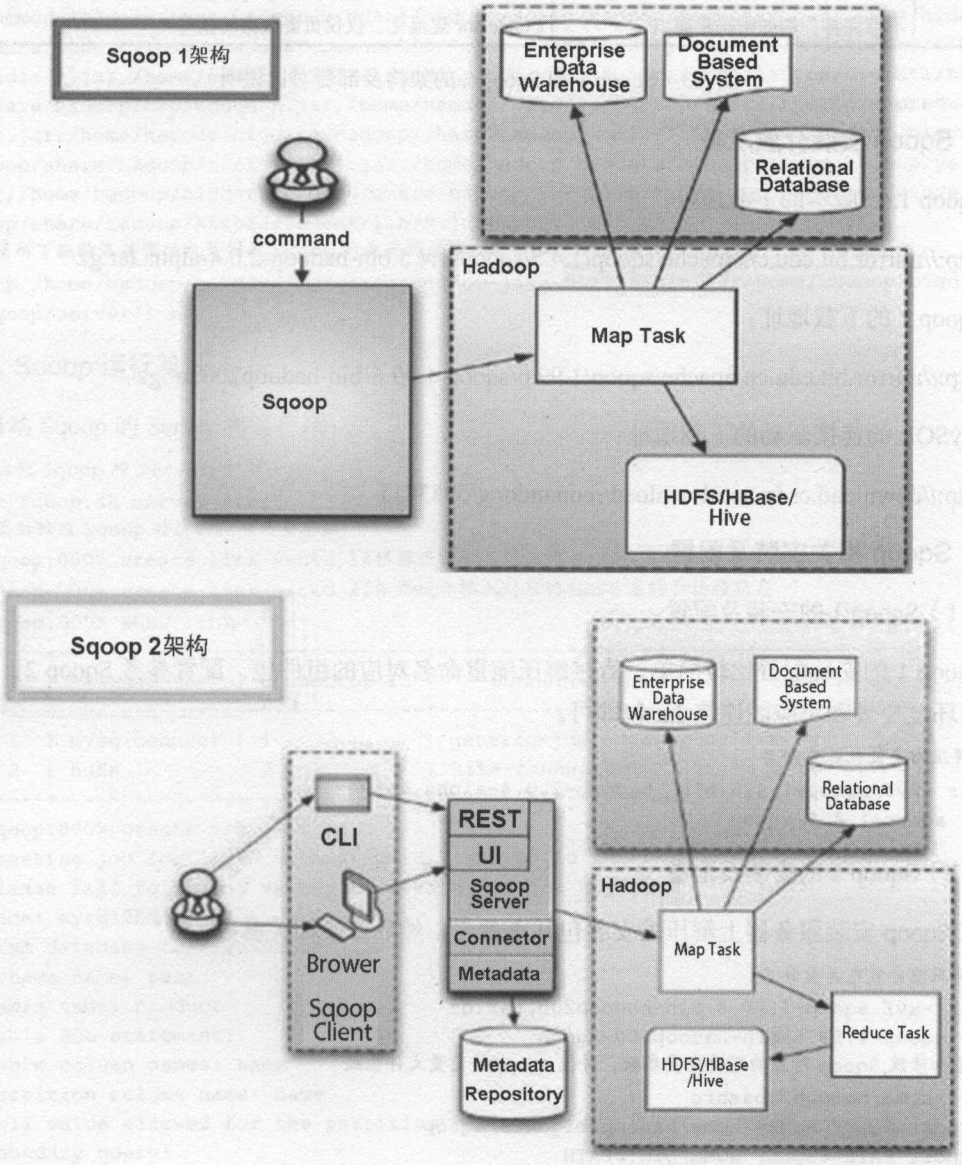
进入 Hive 客户端创建 feigu3 数据库来测试验证（后续建表及加载数据请参考 5.3.1 节“逻辑模型创建”）：

```
hive> create database feigu3;
OK
Time taken: 2.375 seconds
hive> show databases;
OK
default
feigu
feigu3
test
Time taken: 0.059 seconds, Fet
hive>
```

3.2.4 Sqoop 安装及配置

Sqoop 在生产应用中主要有两个版本，即以 1.4.*开头的 Sqoop 1 和以 1.99.*开头的 Sqoop 2。

从选型角度来看，如果大数据平台有统一的安全平台和调度系统，采用 Sqoop1 部署比较简单，对应的 Sqoop 任务直接接入调度系统，也减少了生产系统相关的依赖和运维排查（对于 Hadoop 2.2 以后的版本，Sqoop 1 需要根据具体的版本进行编译）。Sqoop 2 相对于 Sqoop1 增加了安全机制和任务调度管理，如果大数据平台没有统一的安全平台和调度系统，可以基于该 Sqoop 2 形成统一的 Sqoop 数据同步、任务调度接口来运维。Sqoop 1 和 Sqoop 2 的架构及部署对比说明如图 3.3 所示。



比较	Sqoop 1	Sqoop 2
架构	仅仅使用一个 Sqoop 客户端	引入了 Sqoop Server 集中化管理 Connector，以及 REST API、Web。UI，并引入了权限安全机制
部署	部署简单，安装需要 root 权限，Connector 必须符合 JDBC 模型	架构稍复杂，配置部署烦琐
使用	命令行方式，容易出错，格式紧耦合，无法支持所有的数据类型，安全机制不够完善，例如密码暴露	多种交互方式，命令行、Web UI、REST API，Conncetor 集中化管理，所有的 Connector 都安装在 Sqoop Server 上，完善了权限管理机制，Connector 规范化，仅仅负责数据的读写

图 3.3　Sqoop 1 和 Sqoop 2 的架构及部署对比说明

1. Sqoop 安装介质准备

Sqoop 1.4.*版本的下载地址：

http://mirror.bit.edu.cn/apache/sqoop/1.4.5/sqoop-1.4.5.bin-hadoop-2.0.4-alpha.tar.gz

Sqoop 2 的下载地址：

http://mirror.bit.edu.cn/apache/sqoop/1.99.6/sqoop-1.99.6-bin-hadoop200.tar.gz

MySQL 的连接驱动的下载地址：

http://download.csdn.net/download/munandong/5983811

2. Sqoop 相关安装及配置

（1）Sqoop 1 的安装及配置

Sqoop 1 的安装配置比较简单，直接解压缩重命名对应的组件包，配置参考 Sqoop 2，只需设置基本环境变量和对应的连接 Jar 包即可。

```
#解压缩安装包并重命名
tar -xvf sqoop-1.4.5.bin__hadoop-2.0.4-alpha.tar.gz
mv sqoop-1.4.5 sqoop
```

（2）Sqoop 2 的安装及配置

在 Sqoop 安装服务器上解压缩安装包并重命名，然后修改相关参数：

```
#解压缩安装包并重命名
tar -xvf sqoop-1.99.6-bin-hadoop200.tar.gz
mv sqoop-1.99.6-bin-hadoop200 sqoop
#添加修改 Sqoop 对应的环境变量参数，并使 source 变量文件生效
vi /home/hadoop/.bashrc
export SQOOP_HOME=/home/hadoop/bigdata/sqoop
export PATH=$SQOOP_HOME/bin:$PATH
```

```
export CATALINA_HOME=$SQOOP_HOME/server
export LOGDIR=$SQOOP_HOME/logs

#修改 Sqoop 对应的配置参数
vi /home/hadoop/bigdata/sqoop/server/conf/sqoop.properties
org.apache.sqoop.submission.engine.mapreduce.configuration.directory=/home/hadoop/
#修改 Sqoop 对应的配置参数，追加对应的 JAR 包路径
cd /home/hadoop/bigdata/sqoop/server/conf
vi catalina.properties
common.loader=/home/hadoop/bigdata/hadoop/share/hadoop/common/*.jar,/home/hadoop/bi
gdata/hadoop/share/hadoop/common/lib/*.jar,/home/hadoop/bigdata/hadoop/share/hadoop
/hdfs/*.jar,/home/hadoop/bigdata/hadoop/hdfs/lib/*.jar,/home/hadoop/bigdata/hadoop/
share/hadoop/mapreduce/*.jar,/home/hadoop/bigdata/hadoop/share/hadoop/mapreduce/lib
/*.jar,/home/hadoop/bigdata/hadoop/share/hadoop/tools/*.jar,/home/hadoop/bigdata/ha
doop/share/hadoop/tools/lib/*.jar,/home/hadoop/bigdata/hadoop/share/hadoop/yarn/*.j
ar,/home/hadoop/bigdata/hadoop/share/hadoop/yarn/lib/*.jar,/home/hadoop/bigdata/had
oop/share/hadoop/httpfs/tomcat/lib/*.jarexport
#将所下载的连接驱动包复制到 Sqoop 的类库目录下
scp /home/hadoop/bigdata/mysql-connector-java-5.1.26-bin.jar/home/hadoop/bigdata/
sqoop/server/lib/
```

3. Sqoop 运行验证

启动 Sqoop 的 Server 和客户端：

```
#启动 Sqoop 的 Server
sh sqoop.sh server start
#添加修改 Sqoop 对应的环境变量参数
sqoop:000> create link --cid 1#根据提示输入对应的 MySQL 名称和连接信息
sqoop:000> create link --cid 2#根据提示输入对应的 HDFS 名称和连接信息
sqoop:000> show link
+----+--------------+-------------+------------------------+---------+
| Id |     Name     | Connector Id |    Connector Name      | Enabled |
+----+--------------+-------------+------------------------+---------+
| 1  | mysqlconnect | 1           | generic-jdbc-connector | true    |
| 2  | hdfs         | 2           | kite-connector         | true    |
+----+--------------+-------------+------------------------+---------+
sqoop:000> create job -f 1 -t 2 ;
Creating job for links with from id 1 and to id 2
Please fill following values to create new job object
Name: mysql2hdfs1
From database configuration
Schema name: test
Table name: product
Table SQL statement:
Table column names: name
Partition column name: name
Null value allowed for the partition column: true
Boundary query:
```

```
Incremental read
Check column:
Last value:
To Kite Dataset Configuration
Dataset URI: dataset:file:/home/hadoop/sqoop/test
File format:
  0 : CSV
  1 : AVRO
  2 : PARQUET
Choose: 2
Throttling resources
Extractors:
Loaders:
New job was successfully created with validation status OK  and persistent id 1
```

```
sqoop:000> create link --cid 1
Creating link for connector with id 1
Please fill following values to create new link object
Name: mysqlconnect
```

```
Link configuration

JDBC Driver Class: com.mysql.jdbc.Driver
JDBC Connection String: jdbc:mysql://slave01:3306/test
Username: root
Password: ******
JDBC Connection Properties:
There are currently 0 values in the map:
entry#

There were warnings while create or update, but saved successfully.
Warning message: Can't connect to the database with given credentials:
 Access denied for user 'root'@'slave01' (using password: YES)
New link was successfully created with validation status WARNING and p
ersistent id 1
```

```
sqoop:000> create link --cid 2
Creating link for connector with id 2
Please fill following values to create new link object
Name: hdfs

Link Configuration
```

```
HDFS host and port: master:8020/
New link was successfully created with validation status OK and persis
tent id 2
```

```
sqoop:000> show link
+----+-------------+-------------+-----------------------+---------
|
| Id |     Name    | Connector Id |     Connector Name    | Enabled
|
+----+-------------+-------------+-----------------------+---------
|
| 1  | mysqlconnect | 1           | generic-jdbc-connector | true
|
| 2  | hdfs         | 2           | kite-connector         | true
|
+----+-------------+-------------+-----------------------+---------
```

```
sqoop:000> create job -f 1 -t 2
Creating job for links with from id 1 and to id 2
Please fill following values to create new job object
Name: mysql2hdfs

From database configuration

Schema name: test
Table name: product
Table SQL statement:
Table column names:
Partition column name:
Null value allowed for the partition column:
Boundary query:
```

```
sqoop:000> show job
+----+------------+---------------+-------------+---------+
| Id |    Name    | From Connector | To Connector | Enabled |
+----+------------+---------------+-------------+---------+
| 1  | mysql2hdfs1 | 1             | 2           | true    |
+----+------------+---------------+-------------+---------+
```

```
sqoop:000> start job -jid 1
```

相关的 MySQL 表信息如下：

```
mysql> use test
Reading table informati
You can turn off this fe

Database changed
mysql> show tables;
+----------------+
| Tables_in_test |
+----------------+
| product        |
+----------------+
1 row in set (0.00 sec)
```

```
mysql> desc product;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| id    | int(11)     | YES  |     | NULL    |       |
| name  | varchar(20) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
2 rows in set (0.04 sec)
```

3.2.5 Spark 安装及配置

1. Spark 安装介质准备

Spark 的下载地址：

http://spark.apache.org/downloads.html（选择对应的版本和 Hadoop 版本下载安装文件）

http://www.apache.org/dyn/closer.lua/spark/spark-1.5.1/spark-1.5.1-bin-hadoop2.6.tgz

依赖的 Scala 下载地址：

http://www.scala-lang.org/download/2.10.4.html

2. Spark 相关安装及配置

解压缩并重命名所下载的 Scala 及 Spark 安装包：

```
#解压缩并重命名 Scala
tar -zxvf scala-2.10.4.tgz
mv scala-2.10.4 scala
#解压缩并重命名 Spark
tar -zxvf spark-1.5.1-bin-hadoop2.6.tgz
mv spark-1.5.1-bin-hadoop2.6 spark
```

添加对应的系统环境变量：

```
export SCALA_HOME=/home/hadoop/bigdata/scala
export PATH=$PATH:$SCALA_HOME/bin
export SPARK_HOME=/home/hadoop/bigdata/spark
export PATH=$PATH:$SPARK_HOME/bin
```

配置 Spark 配置文件：

```
cd /home/hadoop/bigdata/spark/conf
cp spark-env.sh.template spark-env.sh
#编辑对应的 Spark 环境变量并保存退出
vi spark-env.sh
export SCALA_HOME=/home/hadoop/bigdata/scala
export JAVA_HOME=/usr/java/jdk1.8.0_60
export HADOOP_HOME=/home/hadoop/bigdata/hadoop/
```

```
export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop
SPARK_MASTER_IP=master
SPARK_LOCAL_DIRS=/home/hadoop/bigdata/spark
SPARK_DRIVER_MEMORY=512M
#编辑对应的节点文件并保存退出
cp slaves.template slaves
vi slaves
slave01
slave02
```

3. Spark 运行验证

启动 Spark 并验证相关服务：

```
cd /home/hadoop/bigdata/spark/sbin
sh start-all.sh
```

查看相关服务：

```
[hadoop@master sbin]$ jps
1856 SecondaryNameNode
2578 Jps
1691 NameNode
2043 ResourceManager
2462 Master
[hadoop@master sbin]$
```

```
[hadoop@slave01 ~]$ jps
25777 NodeManager
10035 QuorumPeerMain
8643 Bootstrap
25955 Worker
26037 Jps
5054 RunJar
1231 DataNode
```

3.2.6　Zookeeper 安装及配置

Zookeeper 用来统一系统的状态，在本项目中主要是用来统一 HBase 的状态。

1. Zookeeper 安装介质准备

Zookeeper 的下载地址：

http://mirrors.cnnic.cn/apache/zookeeper/zookeeper-3.4.6/zookeeper-3.4.6.tar.gz

2. Zookeeper 相关安装及配置

首先在 master 节点上进行安装和配置，然后复制至各安装节点：

```
#解压缩安装包并重命名
tar -xvf zookeeper-3.4.6.tar.gz
mv zookeeper-3.4.6zk
#添加修改对应的环境变量参数，并使 source 变量文件生效
vi /home/hadoop/.bashrc
export ZK_HOME=/home/hadoop/bigdata/zk
export PATH=$ZK_HOME/bin:$PATH
source ~/.bashrc
#修改配置文件
cd /home/hadoop/bigdata/zk/conf
cp zoo_sample.cfg zoo.cfg
vi zoo.cfg
dataDir=/home/hadoop/bigdata/zk/zkdata
dataLogDir=/home/hadoop/bigdata/zk/zkdatalog
server.1=master:2888:3888
server.2=slave01:2888:3888
server.3=slave02:2888:3888
#将 master 节点上的安装及配置文件复制到各安装节点
scp .bashrc hadoop@slave01:/home/hadoop/
scp .bashrc hadoop@slave02:/home/hadoop/
scp -r zk hadoop@slave01:/home/hadoop/bigdata/
scp -r zk hadoop@slave02:/home/hadoop/bigdata/
#确认复制到各节点的目录及添加对应的 myid 文件
#将 master 节点上的安装及配置文件复制到各安装节点(master 为 1，与 zoo.cfg 的 Server 一致)
Echo '1'>/home/hadoop/bigdata/myid
```

例如，在 slave02 节点上对应的 myid 如下：

```
[hadoop@slave02 zkdata]$ pwd
/home/hadoop/bigdata/zk/zkdata
[hadoop@slave02 zkdata]$ cat myid
3
[hadoop@slave02 zkdata]$
```

3. Zookeeper 运行验证

```
#在各节点启动 zkServer
cd /home/hadoop/bigdata/zk/bin
./zkServer.sh start
#查看状态
sh zkServer.sh status
```

slave02 为 leader 角色：

```
[hadoop@slave02 bin]$ sh zkServer.sh start
JMX enabled by default
Using config: /home/hadoop/bigdata/zk/bin/../conf/zoo.cfg
Starting zookeeper ... STARTED
[hadoop@slave02 bin]$ jps
1314 NodeManager
1203 DataNode
11507 Jps
11471 QuorumPeerMain
[hadoop@slave02 bin]$ sh zkServer.sh status
JMX enabled by default
Using config: /home/hadoop/bigdata/zk/bin/../conf/zoo.cfg
Mode: leader
```

slave01 为从节点：

```
[hadoop@slave01 bin]$ sh zkServer.sh status
JMX enabled by default
Using config: /home/hadoop/bigdata/zk/bin/../conf/zoo.cfg
Mode: follower
```

3.2.7　HBase 安装及配置

HBase 依赖 Hadoop，Zookeeper 安装完成后进行对应的安装及配置。

1. HBase 安装介质准备

从 HBase 官网下载对应的安装包（选择 1.1.2 版本）：

http://archive.apache.org/dist/hbase/1.1.2/

2. HBase 相关安装及配置

首先在 master 节点上进行安装和配置，然后复制到各安装节点：

```
#解压缩安装包并重命名
tar -xvf hbase-1.1.2-bin.tar.gz
mv hbase-1.1.2-bin hbase
#添加修改对应的环境变量参数，并使 source 变量文件生效
vi /home/hadoop/.bashrc
export HBASE_HOME=/home/hadoop/bigdata/hbase
export PATH=$HBASE_HOME/bin:$PATH
source ~/.bashrc
#修改配置文件
cd /home/hadoop/bigdata/hbase/conf
vi hbase-site.xml
<property>
  <name>hbase.rootdir</name>
  <value>hdfs://master:9000/hbase</value>
</property>
```

```
<property>
  <name>hbase.zookeeper.quorum</name>
  <value>master,slave01,slave02</value>
</property>
<property>
  <name>hbase.cluster.distributed</name>
  <value>true</value>
</property>
<property>
  <name>hbase.zookeeper.property.dataDir</name>
  <value>/home/hadoop/bigdata/zk/zkdata</value>
</property>
#添加对应的 regionserver 服务器名称
vi regionservers
slave01
slave02
#将 master 节点上的安装及配置文件复制到各安装节点
scp .bashrc hadoop@slave01:/home/hadoop/
scp .bashrc hadoop@slave02:/home/hadoop/
scp -r hbase hadoop@slave01:/home/hadoop/bigdata/
scp -r hbase hadoop@slave02:/home/hadoop/bigdata/
```

3. HBase 运行验证

```
#在 master 上启动 HBase
sh $HBASE_HOME/start-hbase.sh
#查看状态
Hbase shell
status
```

```
[hadoop@master bin]$ sh start-hbase.sh
slave02: starting zookeeper, logging to /home/had
s/hbase-hadoop-zookeeper-slave02.out
slave01: starting zookeeper, logging to /home/had
s/hbase-hadoop-zookeeper-slave01.out
master: starting zookeeper, logging to /home/hado
/hbase-hadoop-zookeeper-master.out
starting master, logging to /home/hadoop/bigdata/
op-master-master.out
```

```
[hadoop@master bin]$ jps
26528 Jps
25810 QuorumPeerMain
1590 SecondaryNameNode
1494 RunJar
1751 ResourceManager
1399 NameNode
26424 HMaster
```

```
[hadoop@master bin]$ hbase shell
```

```
hbase(main):001:0> status
2 servers, 0 dead, 4.5000 average load

hbase(main):002:0>
```

3.3　自动化安装及部署说明

在企业生产环境下大数据集群稳定运行至少应达到成百上千台的规模，因此在实际的产线环境下就需要自动化安装及部署来实现集群的快速扩容，本节就对大数据系统的自动化部署设计及调用逻辑进行介绍。

3.3.1　自动化安装及部署整体架构设计

如图 3.4 所示是安装及部署一套基本的大数据系统的步骤和对应的功能脚本说明，其中左边是对应的步骤说明，右边是对应的脚本实现功能说明。

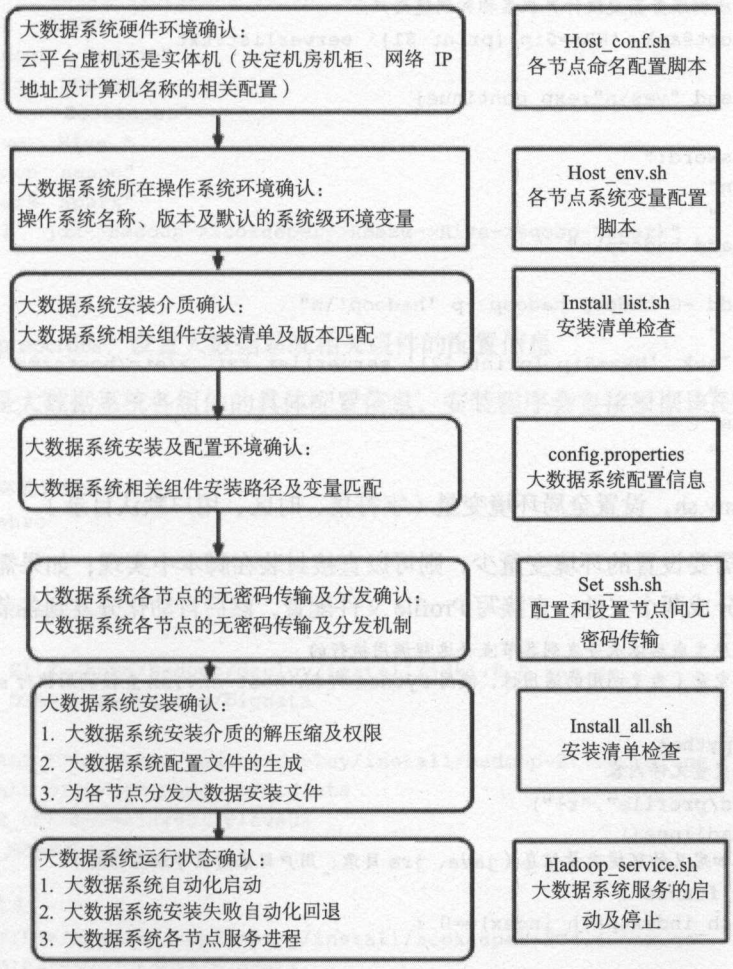


图 3.4　安装及部署一套基本的大数据系统的步骤及功能脚本说明

3.3.2 大数据系统自动化部署逻辑调用关系

由于整套脚本内容比较繁多，这里就不详细列出了，本节只对脚本功能用伪代码进行描述，重点理清大数据系统自动化部署需要使用的脚本作用，功能及其之间的逻辑调用关系（关于详细脚本及部署，欢迎加入飞谷云大数据自动化部署公益项目进行实践了解，www.feiguyun.com）。

（1）Host_conf.sh，各节点命名配置脚本

根据具体环境的不同，该脚本可以被 ansible、cobber、云平台节点管理组件对应的功能实现。

```
#以下只是提取脚本简化的伪代码
#按机器 IP 清单循环执行
For ip in {cat serverlist.txt}
#交互式连接到远程服务器更改计算机名称和创建用户
Spawn Ssh root@awk  'NR==$ip {print $1}' serverlist.txt
expect {
"yes/no" {send "yes\n";exp_continue}
}
expect "password:"
send "$pwd\n"
expect "*]# "
send "groupadd hadoop\n"
expect "*]# "
send "useradd -G hadoop hadoop -p 'hadoop'\n"
Expect "*]# "
Send "Echo `awk  'NR==$ip {print $2}' serverlist.txt  >/etc/hostname \n"
Expect "*]# "
send "exit\n"
expect "*]$ "
```

（2）Host_env.sh，设置全局环境变量（字符集、时区、用户默认目录）

该脚本如果需要设置的环境变量少，则可以直接封装在脚本中实现；如果需要设置的全局变量多，则通常会分成两个文件，直接写 Profile 文件配置，然后自动化分发到各节点来实现。

```
#以下是通过程序生成后依次分发到各节点并远程调用执行的
#配置系统全局变量（为了调用的通用性，使用 Python 脚本，Host_env.sh 直接调用执行 set_profile.py 脚本）
#!/usr/bin/python
#读取当前环境变量文件内容
f=open("/etc/profile","r+")
ftable=f.readlines()
#逐行读取并追加定义的环境变量信息（java、jre 目录、用户目录时区等配置信息）
for line in ftable :
if (classpath_index&path_index)==0 :

changedtable=ftable[:unset_index]+[insert_javahome,insert_jrehome,insert_classpath,
insert_path]+ftable[unset_index:]
```

```
        f.seek(0)
        f.writelines(changedtable)
…
f.close()
```

（3）Install_list.sh，安装清单检查

该脚本主要用于交互显示确认安装的大数据系统组件。

```
#依次显示需要安装的大数据系统清单
bin=`echo $(cd "$(dirname "$0")"; pwd)`
print_usage()
{
  echo "Usage: install [COMPONENT]"
  echo "        where COMPONENT is one of:"
  echo "  jdk   JAVA"
  echo "  hadoop   Hadoop"
  echo "  hbase  Hbase"
  echo "  zk     Zookeeper"
  echo "  hive   Hive "
  echo "  sqoop  sqoop"
  echo "  spark  spark"
  echo "  all   jdk->Hadoop->Zookeeper->HBase->Hive->sqoop->spark"
  echo ""
}
```

（4）config.properties，设置大数据系统相关组件的配置信息

该文件主要是大数据系统各组件的具体配置信息，安装程序会直接根据该配置进行相关目录的安装及设置。

```
#系统所使用的环境变量文件定义
PROFILE=.bashrc
#安装日志定义
LOG_DIR=log
LOG_FILE=bigdata.log
#JDK 定义
JDK_INSTALL_FILE=/home/hadoop/deploy/install/jdk1.8.0.tar.gz
JDK_INSTALL_DIR=/home/hadoop/bigdata
#Hadoop 定义
HADOOP_INSTALL_FILE=/home/hadoop/deploy/install/hadoop-2.7.0.tar.gz
HADOOP_INSTALL_DIR=/home/hadoop/bigdata
HADOOP_SLAVE_SERVERS=slave01,slave02
HADOOP_USER_NAME=hadoop
…
#Zookeeper 定义
ZK_INSTALL_FILE=/home/hadoop/deploy/install/zookeeper-3.4.6.tar.gz
ZK_INSTALL_DIR=/home/hadoop/bigdata
ZK_DATA_DIR=/home/hadoop/opt/bigdata/data/zk
```

```
ZK_LOG_DIR=/home/hadoop/opt/bigdata/logs/zk/logs
ZK_TICK_TIME=180000
CLIENTPORT=2181
ZK_MY_ID=1
…
#HBase 定义
HBASE_INSTALL_FILE=/home/hadoop/deploy/install/hbase-1.1.2-bin.tar.gz
HBASE_INSTALL_DIR=/home/hadoop/bigdata
HBASE_DATA_DIR=/home/hadoop/opt/bigdata/data/hbase
HBASE_HDFS_SERVER=master:49100
HBASE_TICK_TIME=180000
…
#Hive 定义
HIVE_INSTALL_FILE=/home/hadoop/deploy/install/apache-hive-1.2.1-bin.tar.gz
HIVE_INSTALL_DIR=/home/hadoop/bigdata
HIVE_DATA_DIR=/home/hadoop/opt/bigdata/data/hive
HIVE_LOG_DIR=/home/hadoop/opt/bigdata/logs/hive/logs
HIVE_DB_IP=slave03
…
#Spark 定义
SPARK_INSTALL_FILE=…
```

（5）Set_ssh.sh，配置和设置节点间无密码传输

该脚本主要用于部署系统各节点间无密码传输调用，即主节点安装配置完成后的分发和主从节点间远程程序直接调用。

```
#以下是通过程序更改 SSH 服务配置及密钥生成和分发的
#!/usr/bin/python
f=open("/etc/ssh/sshd_config","r+")
ftable=f.readlines()
for line in ftable :
#判断更改 SSH 服务配置信息
    if "RSAAuthentication yes" in line :
        start_index=ftable.index(line)
        ftable[start_index]="RSAAuthentication yes\n"

…
f.seek(0)
f.writelines(ftable)
f.close()
#远程交互连接并生成对应的公私钥文件
spawn ssh -l hadoop $ip
expect {
"yes/no" {send "yes\n";exp_continue}
}
expect "password:"
send "hadoop\n"
expect "*]$ "
```

```
send "ssh-keygen -t dsa -P '' -f ~/.ssh/id_dsa\n"
expect "*]$ "
send "cat ~/.ssh/id_dsa.pub>>~/.ssh/authorized_keys\n"
expect "*]$ "
send "exit\n"
expect "*]$ "
```

（6）Install_all.sh，大数据系统组件安装脚本

该部分脚本主要用于部署大数据各组件的安装及配置。

```
#以下是通过通用命名程序分发和配置安装组件的
install_com()
{
  local name=$1
  cd $bin
  source ~/.bashrc
  if [ -e "$bin/install-${name}.sh" ]; then
    echo "begin to install $name ........."
    . "$bin/install-${name}.sh"
    echo "install $name end."
    echo ""
  fi
  cd $bin
  source ~/.bashrc
  if [ -e "$bin/conf-${name}.sh" ]; then
    echo "begin to config $name............."
    . "$bin/conf-${name}.sh"
    echo "config $name end."
    echo ""
  fi
}

COM=$1
case $COM in
 --help|-help|-h)
   print_usage
   exit
   ;;
 jdk)
   install_com 'jdk'
   ;;
 hadoop)
   install_com 'hadoop'
   ;;
.....
```

其中，install-${name}.sh 以 Hadoop 安装脚本（install-hadoop.sh）细化如下：

```
#获取配置文件中文件路径和安装路径
```

```
HADOOP_INSTALL_FILE=`awk -F "=" '{if($1~/^HADOOP_INSTALL_FILE$/) print $2}' ${config}`
HADOOP_INSTALL_DIR=`awk -F "=" '{if($1~/^HADOOP_INSTALL_DIR$/) print $2}' ${config}`
#将安装文件复制到安装目录下
cp $HADOOP_INSTALL_FILE $HADOOP_INSTALL_DIR
cd $HADOOP_INSTALL_DIR
#解压缩安装包（限于篇幅，检查判断路径及文件是否存在等步骤省略）
tar -xzf $HADOOP_INSTALL_FILE
#对 Hadoop 安装文件赋权
  chmod 755 hadoop
#配置 core（其他项 hdfs、yarn 文件类似，省略）
create_conf_file 'core'
set_out_file "$HADOOP_HOME/conf/core-site.xml.tmp"
tag_start "configuration"
create_property 'fs.defaultFS' "hdfs://${HADOOP_HDFS_SERVER}:49100"
create_property 'io.file.buffer.size' '131072'
create_property 'hadoop.tmp.dir' "${HADOOP_DATA_DIR}/tmp"
tag_end "configuration"
tran_conf_file 'core'
…
#备份配置文件，并将配置好的配置文件更名
tran_conf_file()
if [ -w "$HADOOP_HOME/conf/${file}-site.xml" ];then
    mv"$HADOOP_HOME/conf/${file}-site.xml" "$HADOOP_HOME/conf/${file}-site.xml.bak"
  fi
  mv"$HADOOP_HOME/conf/${file}-site.xml.tmp" "$HADOOP_HOME/conf/${file}-site.xml"
…
#生成从节点配置文件
for slave_server in ${arr[@]}
   do
     echo "${slave_server}" >>$HADOOP_HOME/conf/slaves
  done
  chmod 755 $HADOOP_HOME/conf/slaves
fi
```

Hadoop 安装脚本（install-hadoop.sh）中用到的配置文件项相关函数脚本如下：

```
#创建配置文件
create_conf_file()
{
  local file=$1
  touch "$HADOOP_HOME/conf/${file}-site.xml.tmp"
  if [ $? != 0 ] ; then
    exit 1
  fi
  if [ "${file}" == "yarn" ] ;then
  echo '<?xml version="1.0"?>'>>${HADOOP_HOME}/conf/${file}-site.xml.tmp
  else
echo'<?xmlversion="1.0"
encoding="UTF-8"?>'>>${HADOOP_HOME}/conf/${file}-site.xml.tmp
```

```
echo'<?xml-stylesheettype="text/xsl"
href="configuration.xsl"?>'>>${HADOOP_HOME}/conf/${file}-site.xml.tmp
  fi
}
 #备份配置文件，并将配置好的配置文件更名
tran_conf_file()
if [ -w "$HADOOP_HOME/conf/${file}-site.xml" ];then
   mv"$HADOOP_HOME/conf/${file}-site.xml" "$HADOOP_HOME/conf/${file}-site.xml.bak"
  fi
  mv"$HADOOP_HOME/conf/${file}-site.xml.tmp" "$HADOOP_HOME/conf/${file}-site.xml"
…
#配置 slaves 文件
echo"--------------------------start configure
slaves--------------------------">>${HOME_DIR}/${LOG_DIR}/${LOG_FILE}
if [ -f "${HADOOP_HOME}/conf/slaves" ] ; then
mv ${HADOOP_HOME}/conf/slaves  ${HADOOP_HOME}/conf/slaves.bak
   if [ $? != 0  ] ; then
   echo "cp slaves file Failure!"
   exit 1
   fi
arr=(${HADOOP_SLAVE_SERVERS//,/ })
for slave_server in ${arr[@]}
  do
    echo "${slave_server}" >>$HADOOP_HOME/conf/slaves
  done
  chmod 755 $HADOOP_HOME/conf/slaves
fi
source ${HOME_DIR}/${PROFILE}
echo "--------------------------configure slaves
complete--------------------------">>${HOME_DIR}/${LOG_DIR}/${LOG_FILE}
echo
"*******************************************************************************">>
${HOME_DIR}/${LOG_DIR}/${LOG_FILE}
echo "***************************** hadoop conf
complete*****************************">>${HOME_DIR}/${LOG_DIR}/${LOG_FILE}
```

其中用的几个公用函数脚本如下：

```
#创建配置文件 XML 格式的通用脚本 create_xml.sh
#! /bin/bash
##########################################
##author:  feigu                         #
##date:                                  #
##version: 1.0                           #
##action: xml 配置函数                   #
##########################################
outfile=""
tabs=0
```

```
put(){
    echo '<'${*}'>' >> $outfile
}
#根据配置参数文件将对应的 name 和属性项逐行写入 XML 配置文件
put_tag_txt()
{
   local name=$1
   local text=$2
   echo '<'$name'>'$text'</'$name'>' >> $outfile
}

put_head(){
    put '?'${1}'?'
}

out_tabs(){
    tmp=0
    tabsstr=""
    while [ $tmp -lt $((tabs)) ]
    do
       tabsstr=${tabsstr}'\t'
       tmp=$((tmp+1))
    done
    echo -e -n $tabsstr >> $outfile
}

tag_start(){
    out_tabs
    put $1
    tabs=$((tabs+1))
}

tag_end(){
    tabs=$((tabs-1))
    out_tabs
    put '/'${1}
}

tag_value(){
    out_tabs
    str=""
    str=${1}' value="'${2}'"/'
    put $str
}

tag_text(){
    out_tabs
    local name=$1
```

```
    local text=$2
    put_tag_txt $name $text
}

set_out_file()
{
    outfile="$1"
}
#创建配置文件对应的属性函数
create_property()
{
  local name=$1
  local value=$2
  tag_start "property"
  tag_text "name" $name
  tag_text "value" $value
  tag_end "property"
}
```

3.4　本章小结

对于一套大数据系统，做到自动化部署、自动化监控运维、自动化调度，对于系统的快速实现和稳定运行具有重要的意义。大数据系统的自动化部署包括 XML 格式化脚本、系统环境的配置信息文件、配置文件的生成脚本、安装包的解压缩和复制，通过脚本实现了大数据系统的自动化部署。这样如果产线有多台服务器，只需要更改配置文件，就可以实现多台服务器的自动化部署。当然，部署自动化仅是自动化的一部分，大数据系统的各项服务监控、任务调度及失败重跑等都是可以实现自动化的。

第 4 章
大数据的获取

对数据的获取和处理，是大数据实战项目的重要环节。在 Hadoop 生态系统中，针对不同的应用环节，都有对应的工具或解决方案。接下来的 4 个章节将通过实现一个实际的项目需求，来展示这些工具是如何使用及协调工作的。

目前，国内科技型公司对大数据方面的职位需求是非常旺盛的，很多招聘网站或猎头网站每天都发布有大量的大数据职位信息。我们这个项目的功能，就是对各种招聘网站上关于大数据的职位信息进行采集和落地，对这些数据按照不同的维度进行汇总，统计出结果用以展示，并提供查询界面让用户进行职位检索。

为了实现上述功能，我们对整个项目（飞谷示例项目）做了功能模块划分，展示出数据流走向，如图 4.1 所示。

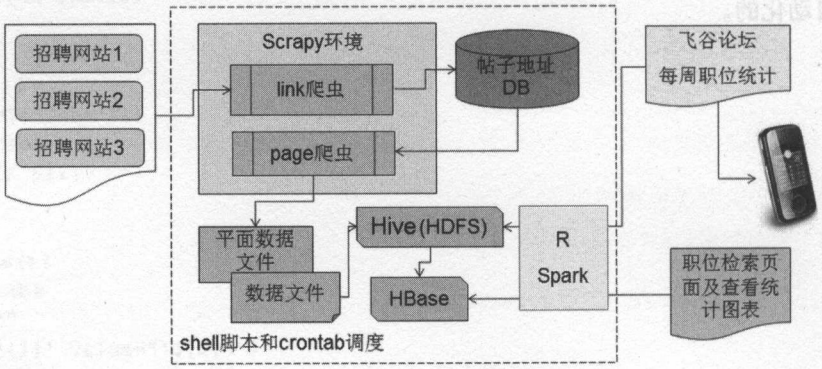


图 4.1　飞谷示例项目功能模块划分图

数据处理过程分为如下 4 个步骤：

① 使用爬虫抓取网络上的招聘职位信息。

② 将数据导入 HDFS，使用 Hive 工具对数据进行清洗转换。

③ 将 Hive 中的数据导入 HBase。

④ 使用 Java API、Spark、R 等技术来操作 HBase 和 Hive 中的数据。

4.1　使用爬虫获取互联网数据

网络爬虫是通过一段程序代码来访问某个 URL 地址，解析其返回的 HTML 文本字符串的技术。因为其实现简单、获取信息廉价、数据时效性强等特点，被广泛应用在搜索引擎网页采集和数据挖掘、监测及自动化测试中。大家比较熟悉的比价网和 12306 抢票插件，也使用了爬虫技术。

最简单的网络爬虫就是 Linux 系统中的 wget 命令，还可以通过安装 curl.so 模块来增强其功能。通过代码来实现爬虫，方法就更多了。很多高级语言都有类似的访问 Web 服务的模块，比如 Java 语言的 java.net 包中的 URLConnection 类、PHP 语言的 file('http://www.example.com/')函数、Python 语言的 urllib.urlopen('http://www.example.com/')方法等。

除了简单的函数之外，还有很多个人或团体开发的免费的爬虫框架可供使用，用以增强原有函数的功能，比如 PHP 语言的 Simple HTML DOM 解析器和 Python 语言的 Scrapy 框架等。不同的爬虫框架其实现原理是类似的，差别在于功能强弱、效率、扩展性和解析 HTML 代码的方式不同。不同的企业可以根据开发团队的技术储备和维护成本，选择不同的爬虫框架。

4.2　Python 和 Scrapy 框架的安装

Python 是目前非常流行的程序设计语言。和 shell 脚本相比，虽然它们都以解释方式执行，但 Python 支持面向对象的编程方法，并且有大而全的类库做支撑，非常适合用在系统整合和执行调度的场合。

此外，和 PHP 语法类似，Python 语法简洁明了，摒弃了烦琐的 Java 语法结构。Python 语法的强制缩进要求，提高了代码的可读性。如果读者有 PHP 或 Perl 语言基础，很容易掌握 Python 语法。

Scrapy 是采用 Python 语言开发的一个快速、可扩展的抓取 Web 站点内容的爬虫框架。读者可以从 http://scrapy.org 站点获取最新版本的 Scrapy 代码，截至 2016 年 5 月，其最终版本是 1.0。在飞谷示例项目中使用的 Scrapy 版本是 0.14.4。下面介绍 Scrapy 的安装步骤。笔者的实验环境是 Ubuntu 操作系统，由于 Scrapy 爬虫运行依赖于一系列的 Python 模块，一些安装介质是通过 apt-get 方式获取的。

（1）安装 setuptools-0.6c11

```
feigu@slave04:~/install_files/setuptools-0.6c11$ pwd
/home/feigu/install_files/setuptools-0.6c11
feigu@slave04:~/install_files/setuptools-0.6c11$ python setup.py install
```

注意：在上述安装步骤中，可能会提示错误信息“Permission denied: '/usr/local/lib/python2.7/dist-packages/test-easy-install-11444.pth'”，原因是由于当前用户没有权限，可以使用 sudo 来执行安装命令。

（2）安装 zope.interface-4.0.1

```
feigu@slave04:~/install_files/zope.interface-4.0.1$ pwd
/home/feigu/install_files/zope.interface-4.0.1
feigu@slave04:~/install_files/zope.interface-4.0.1$ python setup.py install
```

（3）以 root 身份安装 Python 2.7 开发版本

```
root@slave04:/home/feigu# apt-get install python-dev
```

（4）安装 Twisted-12.1.0

```
feigu@slave04:~/install_files/Twisted-12.1.0$ pwd
/home/feigu/install_files/Twisted-12.1.0
feigu@slave04:~/install_files/Twisted-12.1.0$ python setup.py install
```

（5）安装 w3lib-1.2

```
feigu@slave04:~/install_files/w3lib-1.2$ pwd
/home/feigu/install_files/w3lib-1.2
feigu@slave04:~/install_files/w3lib-1.2$ python setup.py install
```

（6）以 root 身份用 apt-get 安装 libxml2 和 libxml2-dev

```
root@slave04:/home/feigu# apt-get install libxml2 libxml2-dev
```

（7）用 apt-get 安装 libxslt1-dev 和 libxslt-dev

```
root@slave04:/home/feigu# apt-get install libxslt1-dev libxslt-dev
```

（8）安装 mysqldb 模块

因为在抓取过程中，需要用到 MySQL 数据库做支撑，所以需要安装 Python 连接 MySQL 的模块。

```
root@slave04:/home/feigu# apt-get install python2.7-mysqldb
```

（9）安装 Scrapy 0.14.4

```
feigu@slave04:~/install_files/Scrapy-0.14.4$ pwd
/home/feigu/install_files/Scrapy-0.14.4
feigu@slave04:~/install_files/Scrapy-0.14.4$ python setup.py install
```

（10）验证安装结果

由于 Scrapy 的运行需要很多 Python 包的支持，以上步骤 1~9 安装完毕后，可以进入 Python 交互界面，查看 lxml 和 OpenSSL 库是否正确安装。操作如下：

```
feigu@slave04:~$ python
Python 2.7.6 (default, Mar 22 2014, 22:59:56)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import lxml
>>> import OpenSSL
>>>
```

输入“import lxml”命令后，如果没有任何提示信息，就说明该模块已经正常安装。

此外，安装 Scrapy 插件后，也可以用命令行来确认 Scrapy 的版本。操作如下：

```
feigu@slave04:~$ whereis scrapy
scrapy: /usr/local/bin/scrapy
feigu@slave04:~$ scrapy version
Scrapy 0.14.4
```

由于 lib 包之间有相互依赖关系，以上 10 个操作步骤要按照次序执行。如果读者的操作系统或软件版本有差异，以上步骤执行时会出现各种提示或错误信息，也属正常现象。需要根据具体信息来安装依赖的 lib 包。

4.3　抓取和解析招聘职位信息

Scrapy 中的爬虫是属于某个项目的。首先可以通过在命令行输入“scrapy startproject <project_name>”命令来创建一个空的项目。操作如下：

```
feigu@slave04:~$ scrapy startproject my_project
feigu@slave04:~$ ls my_project/
                 scrapy.cfg
```

该命令会在硬盘上创建文件夹 my_project，并自动生成子文件夹 my_project 和一些文件，树状结构如下：

```
T:.
|  scrapy.cfg
|
└─my_project
    |  items.py
    |  pipelines.py
    |  settings.py
    |  __init__.py
    |
    └─spiders
            __init__.py
```

每个文件的作用如下：

- scrapy.cfg 是爬虫执行的入口文件。当输入“scrapy crawl”命令让爬虫开始工作时，首先会读取该文件中的配置项内容。
- my_project/items.py 文件定义了爬虫抓取下来的数据，是以何种组织方式存储信息的。比如爬虫抓取的结果可以是标题字符串，也可以是结构化的 JSON 对象，或者是一张图片对应的字节流，items.py 就是用来定义结构化对象中的属性名。
- my_project/pipelines.py 文件定义了信息的保存方式。爬虫抓取的内容，存放在内存对象中，如何保存这些信息，用户可以定义多种方式，比如写入文件、存入 DB 或者直接在控制台输出。Scrapy 会采用管道（pipeline）方式，把内存中的信息依次交给每个管道文件。
- my_project/settings.py 文件保存了爬虫运行时所依赖的配置信息。比如用户定义了两个 pipelines.py 文件，希望把抓取的内容先写入 DB，再输出到控制台，那么就可以在 setttings.py 文件中定义 ITEM_PIPELINES 属性，其值分别给出了两个管道文件的文件名。比如：

```
ITEM_PIPELINES={'page.FilePipelines.PagePipeline':1, 'page.DBPipelines.DBPipeline':2}
```

- my_project/spiders/ 是用来存放具体爬虫文件的目录。爬虫文件需要用户手动创建，在该目录下可以同时存在多个爬虫文件。

一个结构最简单的 Scrapy 爬虫的执行过程如图 4.2 所示。

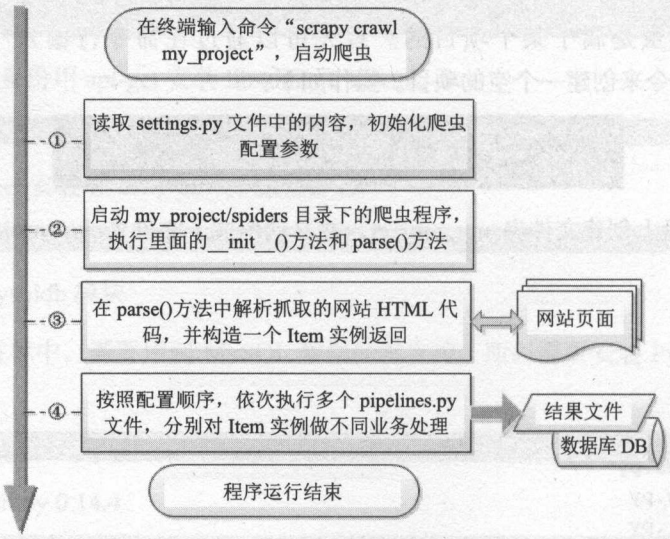


图 4.2　Scrapy 爬虫的执行过程

基于以上执行过程，下面分步骤讲解如何实现抓取和解析职位信息。

（1）创建爬虫文件

使用 Scrapy 可以快速构建一个爬虫系统，其中很方便的一点就是可以直接继承已有的父类 BaseSpider，只需填充里面既有方法的方法体即可。

在 my_project/spiders/下创建 Python 文件 my_spider.py，内容如下：

```
from scrapy.spider import BaseSpider

class MySpider(BaseSpider):
    name = "jobsite1"
    allowed_domains = ["job.jobsite1.com"]
    start_urls = [
        "http://job.jobsite1.com/320_3203995"
    ]

    def parse(self, response):
```

父类 BaseSpider 在 scrapy.spider 包下，父类中有三处地方必须有内容。

- name 属性：用来给爬虫起名字。用户可以在 spiders 目录下创建多个爬虫文件，但必须保证每个文件中的 name 属性值是唯一的，它也是运行“scrapy crawl”命令时所要提供的参数值。
- start_urls 属性：指明了爬虫要抓取的 URL 地址，类型是字符串数组，可以给爬虫指定一个抓取列表。
- parse 方法：是对抓取得到的内容进行解析的方法。参数中的 response 是抓取后获得的内容，对于大多数的 URL 而言，response 值就是这个页面的 HTML 源码字符串。

（2）填充 parse 方法中的代码

在 parse 方法的输入参数 response 对象中，封装了爬虫访问 URL 后得到的响应内容。在 response.body 属性中，存放了 URL 页面所对应的 HTML 源码。无论使用何种语言抓取下来的内容都是字符串文本类型，parse 方法中的代码就是用来解析这个文本字符串的。

比如，我们想从返回的文本中得到职位发布日期、职位名称、公司名称、职位描述等信息，可以采用 Python 自带的字符串处理函数来截取，但此种方法过于烦琐，也不利于代码维护。因为 HTML 文本本身是带有格式和样式的，属于结构化 DOM 对象，所以可以利用里面的特定标签来获取标签内的文本。

Scrapy 使用一种 XPath Selector 机制来解析 HTML 中的数据，它是基于 XPath 表达式语法的（有关 XPath 的详细内容，可参见 http://www.w3.org/TR/xpath/）。XPath Selector 在 Scrapy 中内

置了两种创建方式，分别是 HtmlXPathSelector 和 XmlPathSelector。下面代码展示了如何创建和使用 HtmlXPathSelector。

```
from scrapy.spider import BaseSpider
from scrapy.selector import HtmlXPathSelector

class MySpider(BaseSpider):

def parse(self, response):
hxs = HtmlXPathSelector(response)

job_name=hxs.select('//div[@class="title-info"]/h1/text()').extract()[0].encode('ut
f8')

company_name=hxs.select(//div[@class="title-info  "]/h3/text()').extract()[0].encode
('utf8')
   print job_name
   print company_name
```

通过分析爬虫抓取的地址（以“http://job.liepin.com/320_3203995”为例），可以看出职位名称嵌套在 class="title-info "的 div 当中：

```
……
<div class="title-info">
<h1 title="Hadoop 工程师">Hadoop 工程师</h1>
<h3>上海育碧电脑软件有限公司</h3>
<span class="title-triangle"></span>
</div>
……
```

在解析时就使用 hxs.select('//div[@class="title-info "]/h1/text()')语法来获取内容。由于 select() 方法返回的可能是一个数组，就选择第一个元素返回，encode('utf8')是把获取的字符串转码成 UTF-8 字符集。

（3）测试运行第一个示例爬虫

parse 方法编写完成后，退回到 scrapy.cfg 文件所在的目录，输入“scrapy crawl liepin ”命令，就启动爬虫开始工作了。

```
feigu@slave04:~/my_project$ scrapy crawl liepin
2015-06-03 17:48:08+0800 [scrapy] INFO: Scrapy 0.14.4 started (bot: my_project)
2015-06-03 17:48:08+0800 [scrapy] DEBUG: Enabled extensions: LogStats, TelnetConsole, CloseSpider, WebService, CoreSt
ats, MemoryUsage, SpiderState
2015-06-03 17:48:08+0800 [scrapy] DEBUG: Enabled downloader middlewares: HttpAuthMiddleware, DownloadTimeoutMiddlewar
e, UserAgentMiddleware, RetryMiddleware, DefaultHeadersMiddleware, RedirectMiddleware, CookiesMiddleware, HttpCompres
sionMiddleware, ChunkedTransferMiddleware, DownloaderStats
2015-06-03 17:48:08+0800 [scrapy] DEBUG: Enabled spider middlewares: HttpErrorMiddleware, OffsiteMiddleware, RefererM
iddleware, UrlLengthMiddleware, DepthMiddleware
2015-06-03 17:48:08+0800 [scrapy] DEBUG: Enabled item pipelines:
2015-06-03 17:48:08+0800 [liepin] INFO: Spider opened
2015-06-03 17:48:08+0800 [liepin] INFO: Crawled 0 pages (at 0 pages/min), scraped 0 items (at 0 items/min)
2015-06-03 17:48:08+0800 [scrapy] DEBUG: Telnet console listening on 0.0.0.0:6023
2015-06-03 17:48:08+0800 [scrapy] DEBUG: Web service listening on 0.0.0.0:6080
2015-06-03 17:48:08+0800 [liepin] DEBUG: Crawled (200) <GET http://job.liepin.com/320_320/> (referer: None)
Hadoop工程师
上海育碧电脑软件有限公司
2015-06-03 17:48:08+0800 [liepin] INFO: Closing spider (finished)
2015-06-03 17:48:08+0800 [liepin] INFO: Dumping spider stats:
        {'downloader/request_bytes': 211,
         'downloader/request_count': 1,
         'downloader/request_method_count/GET': 1,
         'downloader/response_bytes': 9089,
         'downloader/response_count': 1,
         'downloader/response_status_count/200': 1,
         'finish_reason': 'finished',
         'finish_time': datetime.datetime(2015, 6, 3, 9, 48, 8, 983998),
         'scheduler/memory_enqueued': 1,
```

Scrapy 在运行时会打印出日志信息，以便监控和排查错误。

4.4　职位信息的落地

前面例子只是简单打印出了职位名称和公司名称，如果要得到页面上完整的职位信息并保存下来，就要分别对 items.py 和 pipelines.py 进行改造。

（1）修改 items.py 文件

我们可以汇总出各类招聘网站职位信息中的共同内容，比如职位名称、公司名称、职位描述、薪资待遇等，定义一些通用字段，放在 items.py 中作为属性值。Scrapy 中定义了 Item 类来存放这些属性。代码片段如下：

```
# encoding: utf-8
from scrapy.item import Item, Field

#定义存放帖子内容的类
class MyItem(Item):
    #生成的文件名
    file_id = Field()
    #工作名称
job_name = Field()
    #工作地点
    job_location = Field()
    #职位描述
    job_desc = Field()
    #公司名称
```

```
    company_name = Field()
    #企业介绍
    company_desc = Field()
```

每个属性值都是 Field 类型的。Field 实际上是内置的 Python 字典类型（Python dict），可以存放不同类型的对象值。

（2）修改 my_spider.py 文件中的 parse 方法

在 parse 方法中创建一个 MyItem 类的实例，把抓取到的内容赋值给字典对象，最后返回该 MyItem 实例。

```
from scrapy.spider import BaseSpider
from scrapy.selector import HtmlXPathSelector
from my_project import items

class MySpider(BaseSpider):

def parse(self, response):
hxs = HtmlXPathSelector(response)
    job_name=hxs.select('//div[@class="title-info
"]/h1/text()').extract()[0].encode('utf8')
company_name=hxs.select('//div[@class="title-info"]/h3/text()').extract()[0].encode
('utf8')
    data = items.MyItem()
data['job_name'] = job_name
data['company_name'] = company_name
return data
```

（3）修改 pipelines.py 文件

parse 方法返回的 MyItem 对象，可以在 pipelines.py 文件的 process_item 方法中得到继续处理，把 MyItems 中的属性写入文件或数据库。示例代码如下，把抓取下来的职位名称和公司地址用逗号分隔开，写入 output.txt 文件中。

```
# encoding: utf-8
import sys

class MyProjectPipeline(object):

#把解析后的内容放入文件中
def process_item(self, item, spider):
fname = '/home/feigu/my_project/output.txt'
outfile = open(fname, 'wb')
outfile.write(item['job_name']+','+item['company_name'])
return item
```

（4）修改 settings.py 文件

Scrapy 支持以管道方式把抓取内容依次输出到多个终端，上面 process_item 方法的最后一定要“return item”，这样保证数据在管道中向后传递。用户可以把 pipelines.py 文件复制一份，在 process_item 方法中继续处理。那么，Scrapy 怎么知道一共有多少个管道文件呢？

my_project/settings.py 文件的 ITEM_PIPELINES 属性，其值列出了所有管道文件的文件名，比如{'page.FilePipelines.PagePipeline':1, 'page.DBPipelines.DBPipeline':2}，其中数字 1 和 2 代表两个文件执行的先后顺序；{}中的信息是把 Item 中的内容依次写入文件和数据库中，PagePipeline 和 DBPipeline 是笔者自定义的两个管道文件，分别封装了写入文件和写入数据库的代码。

通过以上4个步骤，可以简单地实现抓取网页内容并落地。但是，MySpider.py 文件中的 start_url 属性的值是硬代码（hard-code），招聘网站每时每刻都在新增很多职位信息，那么如何实现对这些信息的实时抓取，做到一个都不能少，并且每个页面只抓取一次，避免重复劳动？如何让爬虫的运行更接近真实的企业项目需求？

4.5　两个爬虫配合工作

用户浏览招聘网站，一般过程是先输入检索关键字，得到一个查询结果列表，然后选择查看某个具体招聘职位。每个网页内容都来自于一个 URL 地址，Scrapy 的抓取是以 URL 为单位的。因此，可以考虑开发两个爬虫，其中一个抓取查询结果列表中每个职位 URL 网址，把网页地址保存下来；另一个读取这些网址，分别抓取每个网页的内容。两个爬虫分别命名为 link_spider 和 page_spider。其工作过程如图 4.3 所示。

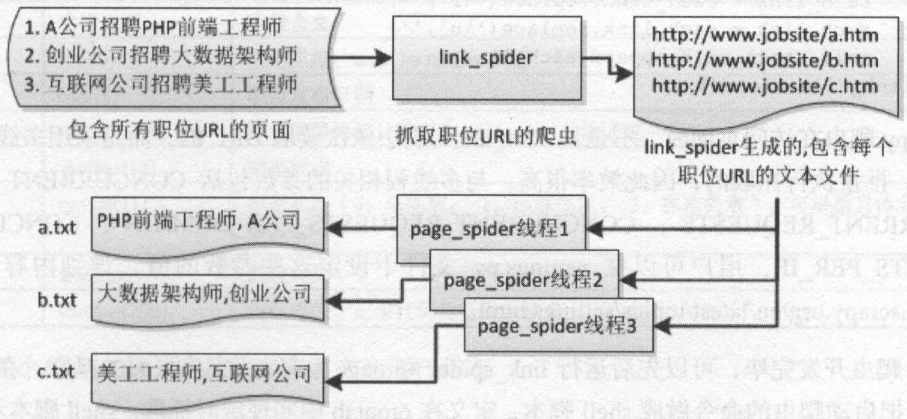


图 4.3　两个爬虫的工作过程

（1）修改 link 爬虫的 link_spider 文件

在 init 方法中，放入要遍历的检索结果地址。代码片段如下：

```
def __init__(self):
  self.start_urls.append("http://www.jobsite.com/search?keyword=JOBKEYWORD")
```

append 方法中的地址参数，要具体分析招聘网站的检索结果 URL 的格式，此处只是示例。有些网站只支持表单 POST 提交方式的请求，那么代码还要模拟一个表单提交才可以。另外，JOBKEYWORD 可用实际的关键字替代，如果有多个关键字，则可以把 start_urls.append 语句放在循环体中。

在 link_spider.py 文件的 parse 方法中，把获得的每个职位 URL 写入文件中。代码片段如下：

```
def parse(self, response):
    hxs = HtmlXPathSelector(response)
    links = hxs.select('//h3[@class="had_news_title"]/a/@href').extract()
    newslink = ''
    for link in links:
        if (len(link) >0):
            newslink = newslink + link + '\n'

    open('../output/link_output/link.txt', 'wb').write(newslink)
```

（2）修改 page 爬虫的 page_spider 文件

在 init 方法中，读取 link 爬虫生成的 link.txt 中的链接地址，作为 page 爬虫的抓取地址。

```
def __init__(self):
    link_file = open('../output/link_output/link.txt', 'r')
    for each_link in link_file:
        each_link = each_link.replace('\r','')
        each_link = each_link.replace('\n','')
        self.start_urls.append(each_link)
    link_file.close()
```

Scrapy 爬虫在访问页面时，不是从 start_url 列表中依次读取 URL 的，而是采用多线程批量读取 URL、批量执行抓取的，因此效率很高。与多线程相关的参数包括 CONCURRENT_ITEMS、CONCURRENT_REQUESTS 、CONCURRENT_REQUESTS_PER_DOMAIN 、CONCURRENT_REQUESTS_PER_IP，用户可以在 settings.py 文件中设定这些参数的值，详细内容可以参考 http://doc.scrapy.org/en/latest/topics/settings.html。

两个爬虫开发完毕，可以先后运行 link_spider 和 page_spider，完成页面的抓取。在实际项目中，可以把启动爬虫的命令做成 shell 脚本，定义在 crontab 中实现定时抓取。shell 脚本示例如下：

```
#!/bin/sh

echo "抓取 job 开始>>>"
```

```
cd /opt/scrapy_job/sites/jobsite1/link
scrapy crawl link >> /opt/scrapy_job/job_scrapy.log
cd /opt/scrapy_job/sites/jobsite1/page
scrapy crawl page >> /opt/scrapy_job/job_scrapy.log
echo "抓取 job 信息结束>>>"
```

以上方法虽然简化了抓取的过程，但是无法保证 link_spider 生成的 link.txt 文件中 URL 的唯一性。比如用户在招聘网站上输入“大数据架构师”和“Hadoop 高级开发”两个关键字，可能会检索到同一条职位信息，也就是说，link.txt 文件中会出现重复的 URL。那么，page 爬虫在 init 方法中读取 link.txt 文件时，必须剔除重复的 URL。

此外，page 爬虫在抓取某个网页时，可能会遇到网络故障无法运行，程序希望能够有机会再次尝试，也就是说，要维护每个 URL 抓取状态。那么，如何解决上述问题，让爬虫架构设计更加合理，适用于生产环境？

4.6　让爬虫的架构设计更加合理

为了解决 URL 重复抓取和维护 URL 的抓取状态，可以引入一个数据库表来替代 link.txt。由于爬虫系统大都是运行在服务器上的自动任务，自动任务最重要的是保证动作状态的可控性和可回溯性，运维人员要能够查看和控制任务的执行状态。

（1）为 link 爬虫抓取 URL 设计数据库表

以 MySQL 数据库为例，设计 job_task 表（见表 4.1）来替代 link.txt。

表 4.1　job_task 表

字段名	字段类型	字段含义
rowid	int(10)	表的主键，auto_increment 类型
jobdate	char(8)	职位发布日期
url	varchar(200)	职位链接地址，设定为 unique key，保证唯一性
webid	varchar(10)	网站标识
curl_status	tinyint(1)	抓取状态（0：未抓取；　1：抓取完成；　2：抓取失败）。可根据具体业务场景来定义
curl_datetime	datetime	抓取时间
fail_reason	varchar(200)	失败原因。记录代码运行异常信息

（2）两个爬虫之间，通过数据库表交互运行状态

修改 link 爬虫的 link_spider.py 文件，替换原来的写入 link.txt 的部分。伪代码如下：

```
def parse(self, response):
    hxs = HtmlXPathSelector(response)
```

```
    links = hxs.select('//h3[@class="had_news_title"]/a/@href').extract()
    newslink = ''
    for link in links:
        if (len(link) >0):
            #此处把链接地址插入 job_task 表，设定抓取状态为 0
            insertJoblink(link,'51job',0)
```

（3）page 爬虫从表中获得所有未抓取的链接地址，在 parse 方法中实现抓取，并修改抓取状态。修改 page_spider.py 文件中的 init 方法和 parse 方法。

```
def __init__(self):
    #从 job_task 表中得到所有未抓取的链接地址
    self.start_urls = getJoblinkByWebid('51job',0)

def  parse(self,response):
    try:
        file_id = response.url.split('/')[-1]
data = items.PageItem()
        #解析职位中的其他要素信息
        ……
        ……
        #更新 job_task 表中的抓取状态
        updateCurlStatus(response.url, 1)
        return data
    catch Exception as e:
        updateCurlStatus(response.url, 2, e)
        print e
```

通过设计 job_task 表，避免了重复抓取的问题，而且使得抓取失败的链接地址可以再次运行。如果 page 爬虫抓取页面时出现了网络异常，经过修复后，只要运行“scrapy crawl page”命令，即可再次尝试抓取。

同样的一条职位信息，其 URL 地址是不变的，即使招聘企业更新了职位信息，变化的也只是招聘日期，URL 地址还是同一个，因此可以使用职位链接表。在有些场景中，需要对同一个 URL 地址进行多次抓取，比如通过爬虫来跟踪商品销量变化，商品 URL 地址一直是同一个，那么爬虫如何来区分内容是否修改过？简单的，可以通过比较两次抓取内容的 MD5 摘要值来实现。

架构的设计是为了解决实际问题，针对一些简单的需求，也没有必要一定使用两个爬虫来完成。抓取二级页面也能在一个爬虫中实现——Spider 类中的 parse 方法可以指定回调函数，在回调函数中解析二级页面的内容，用 yield 方式调用该回调函数。

此外，在实际应用中，有些网站的查询页面要求必须先登录才可以查看，而有些网站是“拒绝”网络爬虫抓取数据的。针对种种状况，Scrapy 也都给出了解决方法，有兴趣的读者可以参阅中文 Scrapy 的文档（http://scrapy-chs.readthedocs.org/zh_CN/0.24/）。

4.7　获取数据的其他方式

大数据项目的数据来源有多种，从数据仓库的角度来划分，可以分为结构化数据和非结构化数据。结构化数据最典型的就是关系数据库表，每条记录中的字段个数、长度、类型都是固定的；非结构化数据就是不方便用数据库二维逻辑来表现的数据，比如文本、图片、各类型的企业报表、音频、视频等；还有一类是半结构化数据，即数据是自描述的，内容和结构混在一起，比如 XML、HTML、JSON 格式对象等。

本章所述的利用爬虫数据抓取信息，其实就是从 HTML 文本中把非结构化数据转化成格式相对整齐的结构化数据的过程。page 爬虫抓取下来的职位信息，保存成了 TXT 文本文件，格式如下：

```
招聘网站标识+分隔符+职位URL+分隔符+职位名称+分隔符+工作地点+分隔符+…+
分隔符+公司网址+分隔符+抓取时间戳
```

分隔符使用 ASCII 码中的 1，在 Python 中表示为 char(1)，生成的文本文件是为了下一步利用 Hive 导入 HDFS，char(1)是 Hive 默认的字段分隔符。

自己编写代码获取数据费时费力，针对一些特定的场景，可以直接使用数据处理工具。Apache Flume 是专门用于将不同服务器上的日志信息、应用程序的输出信息等数据进行汇总，高效导入到 HDFS 中的工具。通过分析 Web 服务器的访问日志，可以挖掘出用户在页面上的停留时间、页面访问次数等信息。Flume 非常适合使用在流式数据挖掘的场景中。

Apache Pig 也是一个导入/处理 HDFS 数据文件的工具。Pig 最早是 Yahoo!公司内部使用的项目，后来贡献给 Apache 组织，目前是 Apache 的顶级项目，版本是 0.15.0。Pig 工具的主要功能并不是数据导入，而是导入后的数据处理能力很强。它提供了 Pig Latin 语法来操作 HDFS 中的数据文件，也被广泛使用在各类大数据项目中。

还有一个专注于数据导入/导出的工具是 Sqoop，它主要适用于把关系数据库中的结构化数据导入/导出 HDFS 文件系统。

4.8　使用 Sqoop 同步论坛中帖子数据

Sqoop 是“SQL to Hadoop”的缩写。Sqoop 最早于 2009 年 5 月成为 Hadoop 的一个贡献模块，2012 年 3 月升级成为 Apache 的顶级项目（http://sqoop.apache.org）。Sqoop 主要用来在 Hadoop 和关系数据库中传递数据。由于 Sqoop 在操作过程中采用了 MapReduce 计算框架，可以实现在集群环境下工作，因此效率非常高。Sqoop 数据处理流程图如图 4.4 所示。

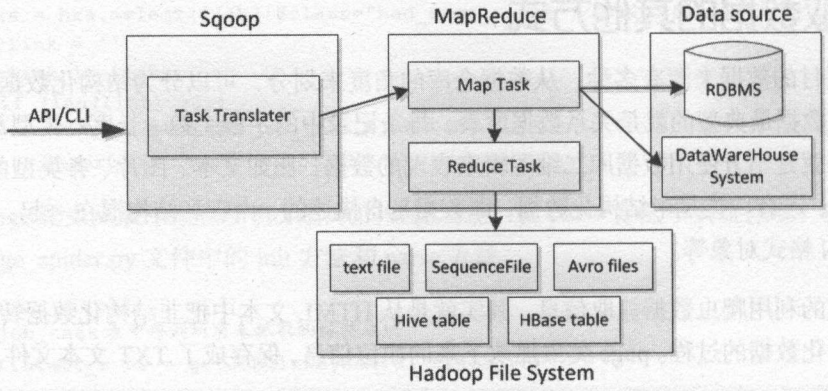


图 4.4　Sqoop 数据处理流程图

Sqoop 数据导入功能采用命令行 CLI（Command Line Interface）方式运行，在使用 JDBC 连接到源数据库后，根据导入后的参数列表，把导入事件打包成一个 MapReduce 任务，提交至 Hadoop 分布式环境，并行地从数据源取数据，生成 HDFS 格式结果文件。Sqoop 生成的结果文件格式可以是文本文件、SequenceFile、AvroFile，Sqoop 也可以把数据直接导入 Hive 或 HBase 中。

Sqoop 中的命令包括 sqoop-import、sqoop-export、sqoop-job、sqoop-merge、sqoop-list-databases、sqoop-list-tables 等，共 14 个，详细内容可以参考官网文档（http://sqoop.apache.org/docs/1.4.5/SqoopUserGuide.html），这里只讲解用途最多的 import 命令的用法。

前面我们从招聘网站上抓取了职位信息，除了职位爬虫之外，还有一些爬虫抓取了很多技术类网站中有关大数据的新闻、帖子，定期插入到飞谷论坛（http://www.feiguyun.com/bbs/forum.php?mod=forumdisplay&fid=42）中。如果想更详细地了解某个招聘公司，查看在哪些帖子里提到了该公司，就需要把这些帖子信息也放入大数据环境中进行检索。飞谷论坛使用 MySQL 数据库，利用 Sqoop 工具，可以把这些帖子信息从 MySQL 导入到 HDFS 文件系统中。

本项目中使用的 Sqoop 版本是 1.4.5，在安装完 Sqoop 后，可以使用“sqoop version”命令查看版本号。用来导出数据的 MySQL 版本是 5.6.15。

把飞谷论坛上 2015 年 5 月份的帖子信息导入到 HDFS 中的命令如下：

```
sqoop import --connect jdbc:mysql://mysqldb:3306/bbs
--username userName  --P
--query'select pid ,subject, REPLACE(REPLACE(message,"\n",""),"\r",""),from_unixtime
(dateline,"%Y-%m-%d %H:%i:%s") from bbs.discuz_forum_post where  fid=42 and (dateline
between unix_timestamp("20150501") and  unix_timestamp("20150531")) and $CONDITIONS '
--split-by pid
--target-dir /user/sqoop_output/201505/
--m 2
```

```
--fields-terminated-by "\0001"
--delete-target-dir
```

上面命令中各参数说明如表 4.2 所示。

表 4.2　参数说明

参数名	含义
connect	连接 RDBMS 的 JDBC 连接字符串[注①]
username	连接数据库的用户名[注②]
P	连接数据库的密码，P 字母大写[注③]
query	提供 SQL 语句从数据库中得到结果集。如果 SQL 语句中有 where 子句，则在 where 最后要加上$CONDITIONS。本例中，从帖子表中取得 2015 年 5 月份发布的所有帖子标题和内容
split-by	指定按照哪个列去分割数据。本例中，pid 是帖子表 discuz_forum_post 的主键[注④]
target-dir	生成的结果文件存放在 HDFS 中的目录名
m	启动 m 个 Map 任务并行运行导入数据
fields-terminated-by	在生成的结果文件中，列之间的分隔符。本例中，使用的"\0001"是 Hive 中表字段的默认分隔符
delete-target-dir	在每次运行导入命令前，如果有就先删除 target-dir 指定的目录

注：① Sqoop 是用 Java 语言开发的，因此只支持 JDBC 方式。在使用 import 命令前，要保证对应的数据库驱动 JAR 包已经放在 Sqoop 的类路径下。

② Sqoop 运行 import 命令时，是把任务分发到集群中的每台 slave 机器上执行的。slave 机器都会去连接数据库，因此要保证每台 slave 机器都有访问数据库的权限。

③ 提供密码有三种方式：P 参数是从控制台输入密码，用*回显屏幕；password 参数后面直接跟密码字符，不安全；password-file 参数后面可以指定密码文件路径，密码事先存放在文件中，适用于非界面交互方式运行的场合。

④ SQL 语句得到的是结果集，MapReduce 运行时是以数据行为单位的，split-by 参指定按照结果集中的哪个列分割成数据行。用来分隔的列通常都是表中的主键。

目前，Apache Sqoop 分为版本 1 和 2，1.4.X 系列是 Sqoop 1，1.99.X 系列是 Sqoop 2，两个版本之间是不兼容的。Sqoop2 版本分为服务器和客户端两个部分，使用 Tomcat 作为 Sqoop Server，服务端可以独立安装 Hadoop 集群的任意一个节点，甚至此节点的 Hadoop 服务可以不启动。在启动服务端后，可以任意安装客户端。在新的架构下，各种命令的执行全部放在了客户端。除此之外，版本 2 的安全性有所提升，有兴趣的读者可以深入研究。

4.9　本章小结

数据埋点、采集、落地是每一个大数据项目的起点。本章通过 Scrapy 爬虫讲解了如何获取扁平化数据，以及使用 Sqoop 来处理结构化数据。前者侧重于代码编写；后者主要是掌握如何配置，因而更适合运维人员使用。而通过解析网站 Web 服务器访问日志来获得数据的方法，在很多类似的书籍中都会介绍，所以本章中并没有涉及此类技术。

第 5 章
大数据的处理

通过爬虫抓取下来的招聘职位数据文件，需要放入 HDFS 中做数据转换、汇总和检索，利用的是 Hive 工具。Hive 是 Hadoop 生态系统中重要的组成部分，主要充当数据仓库建模和 ETL 工具的角色。本章就通过具体的项目需求，来讲解 Hive 中的一些核心技术点。

5.1 Hive 是什么

Apache Hive 是基于 Hadoop 的一个数据仓库工具，可以将结构化的 HDFS 数据文件映射为一个数据库表，并使用 SQL 查询功能来操作其中的数据。用户提交给 Hive 的每个 SQL 语句，都会转换为一个 MapReduce 任务，放在 Hadoop 平台上进行运算，最后把结果集展示在 Hive 中。其优点是学习成本低，可以通过类 SQL 语句快速实现简单的 MapReduce 统计，不必开发专门的 MapReduce 应用，十分适合数据仓库的统计分析。

此外，由于后端有 MapReduce 计算框架做支撑，即使处理 TB 级的数据量，操作的可靠性及性能的扩展性也都有保障。

Hive 最早作为 Facebook 公司的内部项目使用，后来贡献给 Apache，目前是 Apache 的顶级项目，有分别针对 Hadoop 1.x 和 Hadoop 2.x 系列的两个版本。

5.2 为什么使用 Hive 做数据仓库建模

Hadoop 的核心部分包括两个方面：HDFS 和 MapReduce（Yarn）计算框架。HDFS 文件系统保证了数据的完整性、可靠性和容错性；计算框架则提供了一个可平滑扩展的分布式运算环境，这两个特性恰好是一个优秀的数据仓库/挖掘平台所必须具备的。再加上完全免费，且有众多的拥趸，Hadoop 作为数据仓库架构的一种实现方式，已经在很多 IT 企业中得到了深度使用。

由于数据在 Hadoop 中是以 HDFS 格式保存的，如果想实现数据集的汇总、合并、筛选，就只

能使用 MapReduce 算法来实现，而 MapReduce 算法的开发调试又依赖于 Java 语言；虽然 Hadoop 提供了 Streaming 技术，但依然要靠编码来实现，只是换了一种语法而已；并且每一个 MapReduce 任务，在运行时都要提交至 Hadoop 环境，如果出现错误的结果，则只能依赖于 Hadoop 的日志系统来排查，这对于众多做传统数据仓库应用的 DBA 来讲，无疑被拒于千里之外。那么，有没有一种简洁、高效又易于掌握的操作 HDFS 数据的方法呢?

有，答案就是 Hive。Hive 提供了命令行方式来查看 HDFS 中的数据，并能对数据进行检索、按列求和、求平均值或统计条数等功能。Hive 提供了 HiveQL 语法，可以用来创建数据库、创建表、导入数据、检索数据和删除，语法结构类似于 MySQL SQL，它向非编程人员开放了大数据 Hadoop 生态系统。它常被描述为一个构建于 Hadoop 之上的数据仓库基础架构。利用 Hive 工具，数据仓库开发人员在建模和 ETL 数据处理时，更关注于如何设计而更少关心技术实现。

图 5.1 展示了 Hive 和 Hadoop 之间的内部依赖关系。

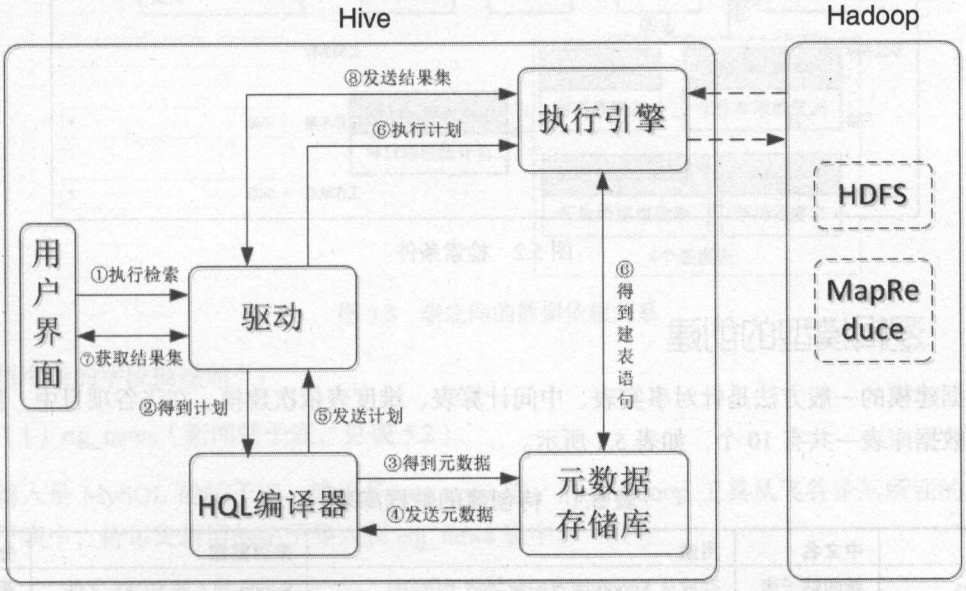


图 5.1　Hive 和 Hadoop 之间的内部依赖关系

（图片摘自 https://cwiki.apache.org/confluence/display/Hive/Design，并有所简化）

5.3　飞谷项目中 Hive 建模步骤

在数据仓库系统中，数据库建模是指按照应用主题，对用户业务数据进行抽象，设计出合理、关联、统一的数据结构的过程。数据库建模可分为三个阶段：概念模型、逻辑模型和物理模型。

数据库建模是数据仓库系统开发的基础性工作，建模结构的优劣将会直接或间接影响到数据的处理效率、性能及扩展性等诸多方面。

虽然使用 Hive 可以很方便地处理 HDFS 中的数据，但是 Hive 本身并不是数据库，它并不存储数据。数据是存放在 Hadoop HDFS 中的，我们用 Hive 可以定义表，让一个表和 HDFS 中的某个文件或目录建立对应关系。数据库中表的定义及表字段描述（元数据）是存放在 Hive 的 metaDB 中的。正如 Hive 的中文意思“蜂巢”那样，使用 Hive 可以非常致密、规则地构造数据仓库系统中的表及其之间的关联关系。

具体到飞谷项目中，我们希望把爬虫抓取的数据最终展示在页面上，针对学历要求、工作地点、薪资待遇及工作年限等维度实现检索和统计，并能够对职位名称、公司名称和招聘日期进行检索，条件如图 5.2 所示。

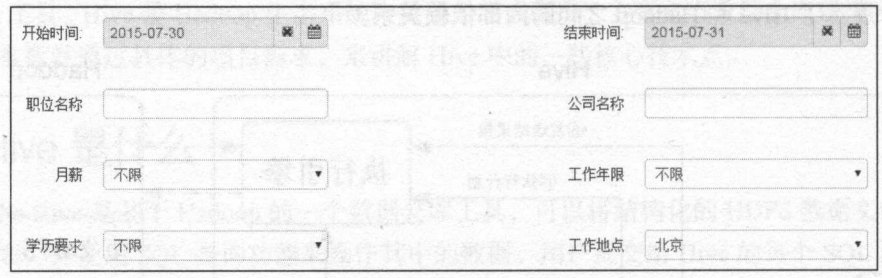


图 5.2　检索条件

5.3.1　逻辑模型的创建

数据建模的一般方法是针对事实表、中间计算表、维度表依次建模。在飞谷项目中，我们要创建的数据库表一共有 10 个，如表 5.1 所示。

表 5.1　待创建的数据库表

表名	中文名	用途	来源数据	分类
stg_news	新闻帖子表	存放从 Sqoop 导入的新闻帖子信息	Sqoop 导入的 HDFS 文件	源表
stg_job	职位表	存放爬虫抓取的原始职位信息	爬虫抓取的文本文件	源表
s_job	职位表	对 stg_job 表中维度数据为空处理后的内容	stg_job	事实表
dm_job	职位表	根据给定条件，对职位信息设定 VIP 标记	s_job	维度表
dim_edu	学历表	学历维度表	s_job	维度表
dim_workyear	工作年限表	工作年限维度表	s_job	维度表
dim_joblocation	职位地域表	职位地域维度表	s_job	维度表

续表

表名	中文名	用途	来源数据	分类
dim_salary	薪资表	薪资维度表	s_job	维度表
daily_dim_sum	每日维度统计表	统计以上 4 个维度的日汇总信息	4 个维度表	维度表
rpt_job	职位表		s_job 及 4 个维度表	聚合表

以上 10 个表之间的数据依赖关系如图 5.3 所示。

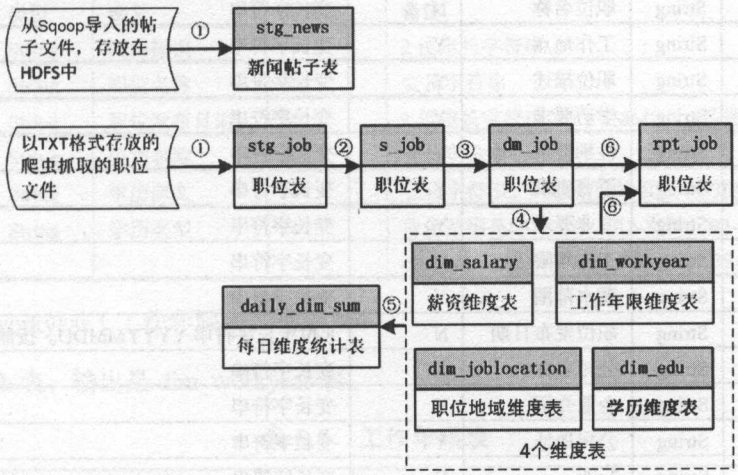


图 5.3　表之间的数据依赖关系

每个表的字段描述如下：

（1）stg_news（新闻帖子表，见表 5.2）

输入是 MySQL 的帖子表，输出是 stg_news 表。利用 Sqoop 工具从飞谷论坛所在的 MySQL 的帖子表中，将每天新增的帖子导入到 stg_news 表中。

表 5.2　新闻帖子表

字段名	类型	含义	可为空	备注
mysql_newsid	String	帖子 ID	N	该帖子在 MySQL 表中的编号
news_title	String	帖子标题	N	变长字符串
content	String	帖子内容	N	变长字符串
create_time	String	帖子在论坛上的发布时间戳	N	yyyy-mm-dd hh24:mi:ss
pt	String	帖子从 MySQL 表中导入到 Hive 的日期	N	8 位定长字符串 YYYYMMDD 作为该表的分区，分区名为 pt

（2）stg_job（职位表，见表 5.3）

输入是爬虫抓取下来的职位 TXT 文件，输出是 stg_job 表。

表 5.3 职位表（stg_job）

字段名	类型	含义	可为空	备注
web_id	String	网站名称	N	变长字符串
web_type	String	招聘网站分类	N	2 位定长字符串。比如 01,02,03,04
job_url	String	职位来源网址	N	变长字符串
job_name	String	职位名称	N	变长字符串
job_location	String	工作地点	Y	变长字符串
job_desc	String	职位描述	Y	变长字符串
edu	String	学历要求	Y	变长字符串
gender	String	性别要求	Y	变长字符串
language	String	语言要求	Y	变长字符串
major	String	专业要求	Y	变长字符串
work_year	String	工作年限	Y	变长字符串
salary	String	薪水范围	Y	变长字符串
job_date	String	职位发布日期	N	8 位定长字符串 YYYYMMDD。按照 job_date 做分区
company_name	String	公司名称	N	变长字符串
company_desc	String	企业介绍	Y	变长字符串
company_address	String	公司地址	Y	变长字符串
company_worktype	String	行业	Y	变长字符串
company_scale	String	规模	Y	变长字符串
company_prop	String	性质	Y	变长字符串
company_website	String	网址	Y	变长字符串
curl_timestamp	String	抓取时刻	N	yyyy-mm-dd hh24:mi:ss

（3）s_job（职位表）

输入是 stg_job 表，输出是 s_job 表。由于 stg_job 表中的学历、薪水、工作年限等内容要抽取维度信息，放入对应的维度表中，为了去掉“噪声”数据，对以上维度为空的情况进行转换，转换后放入 s_job 表中。stg_job 表和 s_job 表的结构完全相同，此处不再重复列出。

（4）dm_job（职位表，见表 5.4）

输入是 s_job 表，输出是 dm_job 表。增加分析属性列，根据 s_job 表中的公司名称是否包含百度、阿里、腾讯进行判断，标注是否是 VIP 职位。dm_job 表和 s_job 表的相同字段不再列出。

表 5.4　职位表（dm_job）

字段名	类型	含义	可为空	备注
vip_flg	String	是否是 VIP 职位	N	1 是 VIP 职位，　0 是普通职位

（5）dim_edu（学历表，见表 5.5）

输入是 s_job 表，输出是 dim_edu 表。

表 5.5　学历表

字段名	类型	含义	可为空	备注
web_type	String	网站标识	N	2 位定长字符串
job_name	String	职位名称	N	变长字符串
pt	String	职位发布日期	N	8 位定长字符串 YYYYMMDD。按照 job_date 做分区
company_name	String	公司名称	N	变长字符串
edu_detail	String	学历原文	N	变长字符串。其值是原始页面中的内容
edu_type	String	学历类型	N	取值范围是[B1-大专、B2-本科、B3-研究生、B9-其他]。dim_edu 表中保存 B1、B2、B3、B9；查询条件展示汉字

（6）dim_workyear（工作年限表，见表 5.6）

输入是 s_job 表，输出是 dim_workyear 表。

表 5.6　工作年限表

字段名	类型	含义	可为空	备注
web_type	String	网站标识	N	2 位定长字符串
job_name	String	职位名称	N	变长字符串
pt	String	职位发布日期	N	8 位定长字符串 YYYYMMDD。按照 job_date 做分区
company_name	String	公司名称	N	变长字符串
workyear_detail	String	工作年限原文	N	变长字符串。其值是原始页面中的内容
workyear_type	String	工作年限类型	N	取值范围是[C1：3 年以下、C2：3 至 5 年、C3：5 年及以上、C9：其他]。dim_edu 表中保存 C1、C2、C3、C9；查询条件展示汉字

（7）dim_joblocation（职位地域表，见表 5.7）

输入是 s_job 表，输出是 dim_joblocation 表。

表 5.7　职位地域表

字段名	类型	含义	可为空	备注
web_type	String	网站标识	N	2 位定长字符串

续表

字段名	类型	含义	可为空	备注
job_name	String	职位名称	N	变长字符串
pt	String	职位发布日期	N	8 位定长字符串 YYYYMMDD。按照 job_date 做分区
company_name	String	公司名称	N	变长字符串
joblocation_detail	String	地域原文	N	变长字符串。其值是原始页面中的内容
joblocation_type	String	地域类型	N	取值范围是[A1-北京、A2-上海、A3-广州、A4-深圳、A9-其他]

（8）dim_salary（薪资表，见表 5.8）

输入是 s_job 表，输出是 dim_salary 表。

表 5.8 薪资表

字段名	类型	含义	可为空	备注
web_type	String	网站标识	N	2 位定长字符串
job_name	String	职位名称	N	变长字符串
pt	String	职位发布日期	N	8 位定长字符串 YYYYMMDD。按照 job_date 做分区
company_name	String	公司名称	N	变长字符串
salary_detail	String	薪资原文	N	变长字符串。其值是原始页面中的内容
salary_type	String	薪资范围	N	月薪取值范围是[D1-一至三万、D2-三至五万、D3-五万以上、D8-面议、D9-其他]

（9）daily_dim_sum（每日维度统计表，见表 5.9）

输入是 rpt_job 表，输出是 daily_dim_sum 表，每天每个 dim_type 一条记录。

表 5.9 每日维度统计表

字段名	类型	含义	可为空	备注
pt	String	职位发布日期	N	8 位定长字符串 YYYYMMDD。按照 job_date 做分区
dim_type	String	维度类型	N	2 位定长字符串。比如 A1,B2,C3
cnt_val	int	汇总值	N	该维度下的合计记录数

（10）rpt_job（职位表，见表 5.10）

输入是 dm_job 表和 4 个维度表，输出是 rpt_job 表。使用 dm_job 左连接每个维度表，生成 rpt_job 表。该结果集导入到 HBase 中供检索使用。

表 5.10　职位表（rpt_job）

字段名	类型	含义	可为空	备注
web_id	String	网站名称	N	变长字符串
web_type	String	招聘网站分类	N	2 位定长字符串
job_url	String	职位来源网址	N	变长字符串
job_name	String	职位名称	N	变长字符串
job_location	String	工作地点	Y	变长字符串
joblocation_type	String	工作地点类型	Y	dim_joblocation.joblocation_type
job_desc	String	职位描述	Y	变长字符串
edu	String	学历要求	Y	变长字符串
edu_type	String	学历要求类型	Y	dim_edu.edu_type
gender	String	性别要求	Y	变长字符串
language	String	语言要求	Y	变长字符串
major	String	专业要求	Y	变长字符串
work_year	String	工作年限	Y	变长字符串
workyear_type	String	工作年限类型	Y	dim_workyear.workyear_type
salary	String	薪水范围	Y	变长字符串
salary_type	String	薪水范围类型	Y	dim_salary.salary_type
job_date	String	职位发布日期	N	8 位定长字符串 YYYYMMDD。按照 job_date 做分区
company_name	String	公司名称	N	变长字符串
company_desc	String	企业介绍	Y	变长字符串
company_address	String	公司地址	Y	变长字符串
company_worktype	String	行业	Y	变长字符串
company_scale	String	规模	Y	变长字符串
company_prop	String	性质	Y	变长字符串
company_website	String	网址	Y	变长字符串
curl_timestamp	String	抓取时刻	N	yyyy-mm-dd hh24:mi:ss
vip_flg	String	是否是 VIP 职位		取值 1 是 VIP 职位，0 是普通职位

5.3.2　物理模型的创建

数据仓库中物理模型的创建，是指根据逻辑模型定义出的表结构，结合具体的数据库类型，转换成符合语法的 DDL 语句的过程。针对前面定义的 10 个表结构，利用 HQL 语句在 Hive 中创建对应的 10 个表。

在创建表之前，通常会先创建数据库。Hive 中数据库的概念，本质上仅仅是表存放的目录或命名空间。使用数据库名称的好处是可以避免表命名冲突，如果用户没有指明具体的数据库，Hive

会使用默认的数据库 default。本项目中数据库名称是 feigu3。创建数据库语法如下：

```
CREATE  DATABASE  feigu3;
```

数据库创建后，Hive 会在 HDFS 文件系统中建立一个对应的目录，数据库中的表将会以这个数据库目录的子目录形式存储。数据库目录的默认位置，在 hive-site.conf 文件中 hive.metastore.warehouse.dir 属性所对应的目录下，例如 feigu3 目录所在的位置是：

```
feigu@slave02:~/scrapy_resume$ hadoop fs -ls /user/hive/warehouse
Found 3 items
drwxr-xr-x   - hadoop supergroup          0 2015-07-19 15:42 /user/hive/warehouse/demo.db
drwxr-xr-x   - feigu  supergroup          0 2015-07-06 20:51 /user/hive/warehouse/feigu3.db
drwxr-xr-x   - hadoop supergroup          0 2015-07-05 00:15 /user/hive/warehouse/test.db
```

数据库的文件目录名是以.db 结尾的。用户也可以在创建数据库时，通过指定 LOCATION 参数来修改这个默认位置。

如果想查看 Hive 中都有哪些数据库目录，可以使用 SHOW DATABASES 命令：

```
hive> SHOW DATABASES;
OK
default
demo
feigu3
test
Time taken: 0.023 seconds, Fetched: 4 row(s)
```

如果想使用哪个数据库，通过 use 命令切换数据库：

```
USE  feigu3;
```

数据库创建完毕，就轮到创建表了，先以 stg_job 表为例，该表用来接收爬虫抓取的原始职位信息。

```
DROP TABLE IF EXISTS stg_job;
CREATE TABLE IF NOT EXISTS stg_job(
    web_id        STRING  COMMENT  'web id',
    web_type      STRING  COMMENT  'website type, 01, 02,...',
    job_url       STRING  COMMENT  'job url',
    job_name          STRING  COMMENT  'job name',
    job_location        STRING  COMMENT  'job location',
    job_desc          STRING  COMMENT  'job desc',
    edu             STRING  COMMENT  'education',
    gender            STRING  COMMENT  'gender',
    language          STRING  COMMENT  'language',
    major             STRING  COMMENT  'major',
    work_year         STRING  COMMENT  'work years',
    salary            STRING  COMMENT  'salary',
    company_name      STRING  COMMENT  'company name',
    company_desc      STRING  COMMENT  'company desc',
    company_address     STRING  COMMENT  'company address',
```

```
    company_worktype    STRING  COMMENT  'company worktype',

    company_scale       STRING  COMMENT  'company scale',
    company_prop        STRING  COMMENT  'company property',
    company_website     STRING  COMMENT  'company website',
    curl_timestamp      STRING  COMMENT  'curl timestamp'
)
COMMENT  'all flat data from webpage'
PARTITIONED BY (
    `pt` STRING  COMMENT  'job post date (format  yyyymmdd)' )
ROW FORMAT DELIMITED
FIELDS TERMINATED BY  '\001'
NULL DEFINED AS  ''
STORED AS TEXTFILE ;
```

对于以上 DDL 语句，需要注意如下几点。

（1）数据类型

Hive 表字段支持的数据类型很丰富，分为基本数据类型和集合数据类型。基本数据类型分别和 Java 的数据类型对应，有 String、Float、Int 等；集合数据类型包括 ARRAY<TYPE　NAME>、MAP<TYPE NAME>、STRUCT<PROPERTY LIST>。

stg_job 表中的字段类型无一例外都采用了 String（字符串），一是因为职位信息中每个属性都比较单一，均为文字描述，不存在结构体；二是为了保证数据内容的完整性，对于 work_year（工作年限）、salary（薪资待遇）等数字类型不做处理，按照字符串类型存储，是什么就放入什么，数据校验、类型转换等工作放在 ETL 环节处理。

（2）键

关系数据库通常使用唯一键和索引来存储数据集，然而 Hive 表中没有传统关系数据库中键的概念，也没有自增键的概念，在 Hive 中可以定义索引来加速某些字段的操作，一个表的索引数据存放在另一个表中，Hive 的索引功能比较有限。避免主键、外键等标准化的主要原因是为了提高检索效率，优化 I/O 性能。但是，非标准化数据可能会导致数据重复或数据不一致。

（3）表分区

上面建表语句中的 PARTITIONED BY 使用了表分区。表分区的概念由来已久，它是用来提升数据存取性能的有效手段，通过使用分区来水平分散压力，将数据从物理上转移到和使用最频繁的用户更接近的地方。

在 Hive 中同样有分区的概念，可以指定表中的多个字段组合在一起，形成分区字段，每个分区中的数据，在 HDFS 中存放在同一个表名下不同的子目录中。选择哪些字段作为分区，要根据

不同的业务场景区别对待。

本例中，职位信息是按天发布的，爬虫也是按天抓取的，用户检索数据时，职位时间段是必不可少的筛选条件，理所当然以日期作为分区字段。PARTITIONED BY(...)括号中的内容是一个分区列表，分区字段的名称可以单独指定（本例中使用 pt），不和表中字段名称一致。此外，按照招聘职位中的工作地点省、市两级组合作为分区，也是一种方案。

除了分区之外，在 Hive 中对表数据分割的方法还有 Bucket（桶），它是比 PARTITION（分区）粒度更细的数据划分方法，可以使特定的查询效率更高。

（4）行和列分隔符

ROW FORMAT DELIMITED 是指定数据行之间的分隔符，该语句必须写在其他子句之前。本例中没有指定行分隔符，完整的语法应该是：

```
ROW FORMAT DELIMITED
LINES TERMINATED BY  '\n'
```

目前，Hive 仅支持回车作为行分隔符，因此 LINES TERMINATED BY 语句可省略。列分隔符使用语法“FIELDS TERMINATED BY '\001'”，字符'\001'是^A 的八进制数，是一个不可见字符。本书 4.7 节中讲到，在 Python 爬虫生成的结果文件中，字段之间是用 char(1)分隔的，就是为了和此处的'\001'对应。

（5）存储方式

建表语句最后出现的 STORED AS TEXTFILE 定义了表文件在 HDFS 中的存放格式，TEXTFILE 是文本格式，这也是默认的文件格式。除此之外，还有 SequenceFile、RCFile 格式，这两种都是以二进制形式存储的，比文本格式的存储占用空间少。stg_job 表中的数据来自于 Python 爬虫生成的文本文件，故只能使用 TextFile 格式。

以下列出创建其他表的 DDL 语句。

（1）stg_news（新闻帖子表）

```
DROP TABLE IF EXISTS stg_news;
CREATE TABLE IF NOT EXISTS stg_news
(
   mysql_newsid   STRING  COMMENT  'mysql thread id, PK',
   news_title      STRING COMMENT 'thread title',
   content        STRING COMMENT 'news content',
   create_time     STRING COMMENT 'thread post timestamp in feigu Dz'
)
COMMENT 'all flat thread from Dz'
PARTITIONED BY (
```

```
    `pt` STRING  COMMENT 'news dump date(format yyyymmdd)' )
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\001'
NULL DEFINED AS ''
STORED AS TEXTFILE;
```

（2）dm_job（职位表）

```
DROP TABLE IF EXISTS dm_job;
CREATE TABLE IF NOT EXISTS dm_job
(
    web_id          STRING  COMMENT  'web id',
    web_type   STRING  COMMENT 'website type, fixed 01, 02',

    job_url     STRING  COMMENT 'job url',
    job_name        STRING  COMMENT 'job name',
    job_location STRING   COMMENT 'job location',
    job_desc  STRING  COMMENT 'job desc',
    edu  STRING  COMMENT 'education',
    gender  STRING  COMMENT 'gender',
    language  STRING  COMMENT 'language',
    major  STRING  COMMENT 'major',
    work_year  STRING  COMMENT 'work years',
    salary  STRING  COMMENT 'salary',
    company_name  STRING  COMMENT 'company name',
    company_desc  STRING  COMMENT 'company desc',
    company_address  STRING  COMMENT 'company address',
    company_worktype  STRING  COMMENT 'company worktype',
    company_scale  STRING  COMMENT 'company scale',
    company_prop  STRING  COMMENT 'company property',
    company_website  STRING  COMMENT 'company website',
    curl_timestamp  STRING  COMMENT 'curl timestamp',
    vip_flg STRING  COMMENT 'is vip jobinfo, 0 or 1'
)
COMMENT 'compute vip flag from s_job'
PARTITIONED BY (
    `pt` STRING  COMMENT 'job post date(format yyyymmdd)' )
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\001'
NULL DEFINED AS ''
STORED AS SEQUENCEFILE;
```

（3）dim_edu（学历表）

```
DROP TABLE IF EXISTS dim_edu;
CREATE TABLE IF NOT EXISTS dim_edu
(
    web_type   STRING  COMMENT 'website type, fixed 01, 02',
    job_name        STRING  COMMENT 'job name',
    company_name  STRING  COMMENT 'company name',
```

```
    edu_detail  STRING  COMMENT 'raw data from webpage edu info',
    edu_type STRING  COMMENT 'edu enum type 01,02,03'
)
COMMENT 'edu dimision info from s_job'
PARTITIONED BY (
    `pt` STRING  COMMENT 'job post date(format yyyymmdd)' )
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\001'
NULL DEFINED AS ''
STORED AS SEQUENCEFILE;
```

（4）dim_workyear（工作年限表）

```
DROP TABLE IF EXISTS dim_workyear;
CREATE TABLE IF NOT EXISTS dim_workyear
(
    web_type   STRING  COMMENT 'website type, fixed 01, 02',
    job_name       STRING  COMMENT 'job name',
    company_name  STRING  COMMENT 'company name',
    workyear_detail  STRING  COMMENT 'raw data from webpage work years info',
    workyear_type STRING  COMMENT 'work year enum type 01,02,03'
)
COMMENT 'work years dimision info from s_job'

PARTITIONED BY (
    `pt` STRING  COMMENT 'job post date(format yyyymmdd)' )
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\001'
NULL DEFINED AS ''
STORED AS SEQUENCEFILE;
```

（5）dim_joblocation（职位地域表）

```
DROP TABLE IF EXISTS dim_joblocation;
CREATE TABLE IF NOT EXISTS dim_joblocation
(
    web_type   STRING  COMMENT 'website type, fixed 01, 02',
    job_name       STRING  COMMENT 'job name',
    company_name  STRING  COMMENT 'company name',
    joblocation_detail  STRING  COMMENT 'raw data from webpage job location info',
    joblocation_type STRING  COMMENT 'job location enum type 01,02,03'
)
COMMENT 'job location dimision info from s_job'
PARTITIONED BY (
    `pt` STRING  COMMENT  'job post date(format yyyymmdd)' )
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\001'
NULL DEFINED AS ''
STORED AS SEQUENCEFILE;
```

（6）dim_salary（薪资表）

```
DROP TABLE IF EXISTS dim_salary;
CREATE TABLE IF NOT EXISTS dim_salary
(
   web_type   STRING  COMMENT 'website type, fixed 01, 02',
   job_name       STRING  COMMENT 'job name',
   company_name  STRING  COMMENT 'company name',
   salary_detail  STRING  COMMENT 'raw data from webpage salary info',
   salary_type STRING  COMMENT 'salary enum type 01,02,03'
)
COMMENT 'job salary dimision info from s_job'
PARTITIONED BY (
   `pt` STRING  COMMENT 'job post date(format yyyymmdd)' )
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\001'
NULL DEFINED AS ''
STORED AS SEQUENCEFILE;
```

（7）rpt_job（职位表）——左连接维度表后的职位表，和 HBase 对应

```
DROP TABLE IF EXISTS rpt_job;
CREATE TABLE IF NOT EXISTS rpt_job
(
   web_id         STRING  COMMENT  'web id',
   web_type   STRING  COMMENT 'website type, fixed 01, 02',
   job_url     STRING  COMMENT 'job url',
   job_name       STRING  COMMENT 'job name',
   job_location STRING   COMMENT 'job location',
   joblocation_type STRING  COMMENT 'dim_joblocation.joblocation_type',
   job_desc  STRING  COMMENT 'job desc',
   edu  STRING  COMMENT 'education',
   edu_type STRING  COMMENT 'dim_edu.edu_type',
   gender  STRING  COMMENT 'gender',
   language  STRING  COMMENT 'language',
   major  STRING  COMMENT 'major',
   work_year  STRING  COMMENT 'work years',
   workyear_type  STRING  COMMENT  'dim_workyear.workyear_type',
   salary  STRING  COMMENT 'salary',
   salary_type STRING  COMMENT 'dim_salary.salary_type',
   company_name  STRING  COMMENT 'company name',
   company_desc  STRING  COMMENT 'company desc',
   company_address  STRING  COMMENT 'company address',
   company_worktype  STRING  COMMENT 'company worktype',
   company_scale  STRING  COMMENT 'company scale',
   company_prop  STRING  COMMENT 'company property',
   company_website  STRING  COMMENT 'company website',
   curl_timestamp  STRING  COMMENT 'curl timestamp',
```

```
    vip_flg STRING  COMMENT 'is vip jobinfo, 0 or 1'
)

COMMENT 'dm_job left join dim_* , the output data dump into hbase'
PARTITIONED BY (
    `pt` STRING  COMMENT 'job post date(format yyyymmdd)' )
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\001'
NULL DEFINED AS ''
STORED AS SEQUENCEFILE;
```

（8）daily_dim_sum（每日维度统计表）

```
DROP TABLE IF EXISTS daily_dim_sum;
CREATE TABLE IF NOT EXISTS daily_dim_sum
(
    dim_type  STRING  COMMENT 'dim types',
    cnt_val  int COMMENT 'count(job_name) group by dim_type'
)
COMMENT 'daily dimisons sum info'
PARTITIONED BY (
    `pt` STRING  COMMENT 'job post date(format yyyymmdd)' )
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\001'
NULL DEFINED AS ''
STORED AS SEQUENCEFILE;
```

5.3.3 将爬虫数据导入 stg_job 表

要执行上面的 10 个创建表脚本，可以登录 Hive 客户端 beeline，或者使用 Hive CLI 界面，把每条 DDL 语句贴进去执行。这样做显然很烦琐，也不具备自动执行的能力。那么，有没有一种简单的方法呢?

在 Hive 中可以使用“-f <文件名>”参数方式从文件中执行 Hive 查询语句。依照惯例，一般把这些查询语句保存为*.q 或*.hql（hive query language）后缀的文件。我们可以把上面的 DDL 语句合并在一个文本文件中，命名为 create-feigu.hql。

```
[hadoop@master ~]$ hive -f /home/hadoop/create-feigu3.hql
OK
Time taken: 3.709 seconds
OK
Time taken: 6.945 seconds
OK
Time taken: 1.505 seconds
OK
Time taken: 1.153 seconds
```

或者，在 Hive shell 中使用 source 命令执行脚本文件：

```
hive> source /home/hadoop/create-feigu3.hql;
OK
Time taken: 1.701 seconds
OK
Time taken: 16.169 seconds
OK
Time taken: 2.215 seconds
OK
Time taken: 1.528 seconds
OK
Time taken: 0.937 seconds
OK
```

脚本执行完成后，可以输入 show tables 命令查看表名：

```
hive> show tables;
OK
daily_dim_sum
dim_edu
dim_joblocation
dim_salary
dim_workyear
dm_job
rpt_job
s_job
stg_job
stg_news
Time taken: 1.387 seconds, Fetched: 10 row(s)
```

前面提到过，Hive 中的每个表在 HDFS 中都对应一个文件目录，可以在 Hive 中输入 Hadoop 的 dfs 命令查看这些目录的位置：

```
hive> dfs -ls /user/hadoop/hive-0.13.0/warehouse/feigu3.db;
Found 10 items
2015-08-03 17:09 /user/hadoop/hive-0.13.0/warehouse/feigu3.db/daily_dim_sum
2015-08-03 17:09 /user/hadoop/hive-0.13.0/warehouse/feigu3.db/dim_edu
2015-08-03 17:09 /user/hadoop/hive-0.13.0/warehouse/feigu3.db/dim_joblocation
2015-08-03 17:09 /user/hadoop/hive-0.13.0/warehouse/feigu3.db/dim_salary
2015-08-03 17:09 /user/hadoop/hive-0.13.0/warehouse/feigu3.db/dim_workyear
2015-08-03 17:09 /user/hadoop/hive-0.13.0/warehouse/feigu3.db/dm_job
2015-08-03 17:09 /user/hadoop/hive-0.13.0/warehouse/feigu3.db/rpt_job
2015-08-03 17:08 /user/hadoop/hive-0.13.0/warehouse/feigu3.db/s_job
2015-08-03 17:08 /user/hadoop/hive-0.13.0/warehouse/feigu3.db/stg_job
2015-08-03 17:08 /user/hadoop/hive-0.13.0/warehouse/feigu3.db/stg_news
```

我们看到，目前 stg_job 表还只是一个文件夹，里面没有任何数据。要把爬虫抓取的数据导入表中，读者可能会很自然想到 INSERT 语句。在 Hive 中没有提供行级别的数据插入、数据更新和数据删除功能，往表中“装载”大量的数据，Hive 提供了两种办法，分别是从外部文件或 HQL 结果集中将数据批量插入至 Hive 表。

LOAD DATA 命令是从文件将数据加载至 Hive 表的。如下命令将爬虫抓取的数据加载至 stg_job 表：

```
LOAD DATA LOCAL INPATH  '/home/feigu/scrapy_output/20150501'
```

```
OVERWRITE INTO TABLE stg_job
PARTITION  (pt='20150501')
```

执行 LOAD DATA 命令的输出如下：

```
hive> LOAD DATA LOCAL INPATH
    > '/home/hadoop/jobdata/daily/20150501'
    > OVERWRITE INTO TABLE stg_job
    > PARTITION (pt='20150501');
Copying data from file:/home/hadoop/jobdata/daily/20150501
Copying file: file:/home/hadoop/jobdata/daily/20150501/liepin1.dat
Copying file: file:/home/hadoop/jobdata/daily/20150501/51job1.dat
Copying file: file:/home/hadoop/jobdata/daily/20150501/dajie1.dat
Loading data to table feigu3.stg_job partition (pt=20150501)
Partition feigu3.stg_job{pt=20150501} stats: [numFiles=3, numRows=0, totalSize=35468, rawDataSize=0]
OK
Time taken: 21.816 seconds
```

执行 LOAD DATA 命令需要注意如下几点。

（1）LOCAL INPATH

要加载的数据文件可以有两种不同的来源：LOCAL INPATH 是从执行 LOAD 命令的本机中的某个目录下加载数据的；没有 LOACL 参数，只有 INPATH 是从 Hadoop 分布式文件系统中的某个目录下加载数据的。INPATH 后面引号中的字符串，既可以是一个目录名，也可以是一个文件名。如果指定目录名，Hive 会尝试加载该目录下的每个文件。

INPATH 后面的字符串，通常都会指明一个目录，这样处理起来更加方便，但是不允许再包含子目录。另外，如果 LOCAL 参数存在，Hive 只会复制文件到对应的 Hive 表目录下；如果 LOCAL 参数不存在，则是把 HDFS 中的一个目录移动到 Hive 表对应的目录下，原有目录和文件会被删除。

（2）OVERWRITE

如果命令中包含 OVERWRITE 关键字，Hive 表中对应目录下已有的数据会被先删除，然后再加载新的数据；如果没有这个关键字，仅会把 INPATH 后面字符串中对应的数据追加到 Hive 表目录下。这就好比是写文件的两种方式：一种是覆盖原有文件内容；一种是在原有文件末尾接着写。

在实际应用场景中，通常 OVERWRITE 和 PARTITION 配合使用，实现对某个分区数据的重复写入。比如爬虫抓取当天数据，由于某个招聘网站出现访问故障，数据没有抓取下来，导致最终导入到 Hive 表中的数据不完整，这时可以通过人工干预的方式，重新抓取完整的数据，再次导入到 Hive 表中相同日期的分区下。

（3）PARTITION（分区）

在创建 stg_job 表时，我们用 pt（职位发布日期）作为该表的分区字段。在使用 LOAD DATA 命令导入数据时，也必须指定要把数据文件放在哪个分区下。（pt='20150501'）的含义是告诉 Hive

表，把 5 月 1 日爬虫抓取生成的职位文件放在 20150501 这个目录下。该目录是隶属于 stg_job 父目录的。由于 Python 爬虫设定的是每天晚上抓取当日的职位，抓取完成后再导入到相同日期的 Hive 分区下，这样就使得职位发布时间和目录对应起来。

命令执行完成后，可以在 HDFS 文件系统中查看到所创建的这个目录：

```
hive> dfs -ls /user/hadoop/hive-0.13.0/warehouse/feigu3.db/stg_job;
Found 1 items
/user/hadoop/hive-0.13.0/warehouse/feigu3.db/stg_job/pt=20150501
```

在 Hive CLI 中，可以通过 SHOW PARTITIONS 命令查看 stg_job 表中的所有分区：

```
hive> SHOW PARTITIONS stg_job;
OK
pt=20150501
Time taken: 0.443 seconds, Fetched: 1 row(s)
```

LOAD 命令的运行通常是很快的，这是因为 Hive 并不会验证用户加载的数据和表的模式是否匹配。Hive 会验证文件格式是否和表结构定义的一致。比如，定义的存储格式是 SEQUENCEFILE，那么加载的文件也必须是相同的格式。

数据加载完成后，就可以通过 SELECT 语句查看特定分区中的数据了：

```
hive> select * from stg_job where pt='20150501';
OK
```

5.4　使用 Hive 进行数据清洗转换

LOAD DATA 命令解决了数据导入问题，数据进入到 Hive 中后，工作才刚刚开始。爬虫抓取下来的招聘数据，可能会存在诸多问题，比如：

- 数据项为空。抓取下来的薪资待遇、工作年限等字段可能为空，空数据会干扰指标的统计结果，必须剔除。
- 检索标准不一致。在设计的 Web 页面中，“月薪”检索项预先设定了几个标准：一至三万、三至五万、五万及以上。由于不同来源的招聘网站，薪资待遇的设定是不一样的，有的是“8000~10000/月”，有的是“年薪 15 万”，那么如何与所设定的三个标准进行匹配，得到正确的检索结果呢？

为了解决上述问题，就需要用到 ETL 技术。

ETL 是数据抽取（Extract）、转换（Transform）、加载（Load）英文的缩写，ETL 技术是构建数据仓库的关键环节，占到整个数据仓库构建工作量的 60%以上；而且，ETL 是数据仓库日常维护中问题最多、最烦琐的一部分。所以，选择一个合适的 ETL 工具就显得尤为重要了。

ETL 处理流程如图 5.4 所示。

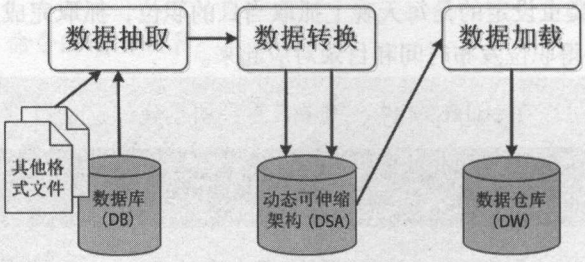


图 5.4　ETL 处理流程图

Hive 作为一个可靠的 ETL 工具，在高效性、扩展性、容错性等方面的表现非常突出。Hive 的每条操作指令几乎都会转换成 MapReduce 任务，利用分布式架构来完成运算，这在效率和扩展性上，都比使用传统的数据库存储过程（Stored Procedure）或某种高级语言更为优越；并且 Hive 面对由于错误数据而导致的程序异常，应对起来更加“从容”，某个节点的程序或硬件发生故障，不会影响整个任务的结果输出。此外，相对于其他收费的 ETL 工具（比如 Informatic ETL），Hive 的开源和免费，一方面，使得用户经验得以在全球范围内积累和传播；另一方面，企业也可以根据需要修改源码来满足更多的应用场景，非常适合于“BAT”类型的公司使用。

5.5　数据清洗转换的必要性

在大多数情况下，原始数据采集完成进入 HDFS 后，都要进行预处理，将原始的输入数据转换成适合分析的数据形式。但是数据并不完美，缺点表现为：

- 数据缺失。比如缺少属性值，缺少重要的统计属性，或者仅包含加总数据。
- 噪声。包含错误或异常值，例如 Salary="-100"。
- 不一致。编码或命名差异，例如品牌=耐克，商品品牌=Nike。

数据质量是数据挖掘质量的前提条件，重复或缺失数据会导致错误的统计结果。数据预处理分为数据清理、数据变换和数据集成，如图 5.5 所示。

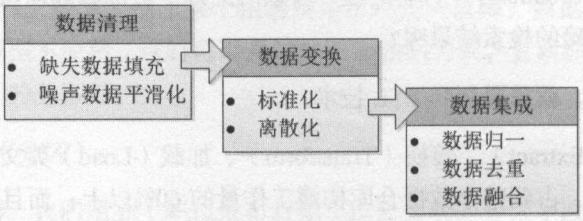


图 5.5　数据预处理示意图

5.6　使用 HiveQL 清洗数据、提取维度信息

5.6.1　使用 HQL 清洗数据

在 5.3.1 节中，逻辑模型中的 stg_job 表和 s_job 表分别用来存放爬虫抓取的原始数据和针对维度信息去空后的数据。数据从 stg_job 表到 s_job 表，使用了 HQL 语句直接导入，在导入的同时，对维度为空的字段进行了转换。HQL 如下：

```
INSERT OVERWRITE TABLE  s_job PARTITION (pt)
SELECT
    web_id,
    web_type,
    job_url,
    job_name,
    CASE
      WHEN job_location IS NULL OR TRIM(job_location) = '' THEN "--"
      ELSE job_location
    END job_location,
    job_desc,
    CASE
      WHEN edu IS NULL OR TRIM(edu) = '' THEN "--"
      ELSE edu
    END edu,
    gender,
    language,
    major,
CASE
      WHEN work_year IS NULL OR TRIM(work_year) = '' THEN "--"
      ELSE  work_year
    END work_year,
CASE
      WHEN salary IS NULL OR TRIM(salary) = '' THEN "--"
      ELSE salary
    END salary,
    company_name,
    company_desc,
    company_address,
    company_worktype,
    company_scale,
    curl_timestamp,
    pt
FROM
  stg_job
WHERE
  pt = '20150501'
```

对于上面的 HQL 要注意以下几点。

（1）INSERT OVERWRITE TABLE …PARTITION (…) SELECT

这是在 Hive 中把一个查询结果集写入另一个表中最常用的句式。在很多 RDBMS 的 SQL 中，也都支持 SELECT…INTO 句式。目标表中的列名是不写的，SELECT…FROM 之间检索出来的字段数目和类型，要和目标表中的保持一致。

PARTITION(pt)在目标表中使用了动态分区——可以基于查询参数推断出需要创建的分区名称，即 stg_job 表中职位发布日期（分区）是'20150501'的数据，经过处理后，导入到目标表 s_job 中同样的分区下。使用这个句式，会在 s_job 表中自动创建分区。

使用 OVERWRITE 关键字会先删除 s_job 表中 pt='20150501'分区下的数据，这样就使得相同分区下的数据可以重复导入了。

（2）CASE…WHEN…THEN…句式

这个句式用于处理单个列的查询结果，可以使用 IF…THEN…ELSE…AS 子句。IF… ELSE 句式只限定了两种非此即彼的情况，如果要根据某列的值匹配多种情况，则可以使用 CASE…WHEN 句式。在上面的 HQL 中，分别对 job_location（工作地点）、edu（学历）、work_year（工作年限）和 salary（薪资范围）四列做了数据为空的转换，是因为这四列分别对应了 4 个维度表。由于对每列的值只做了是否为空的转换，因此此处的 CASE…WHEN…THEN 句式可以用 IF…ELSE 替代。CASE…WHEN 句式比较复杂的写法如下：

```
CASE
    WHEN salary < 5000 THEN  'low'
    WHEN  salary >= 5000 AND salary < 7000 THEN  'middle'
    WHEN  salary >= 7000 AND salary < 10000 THEN  'high'
END AS bracket
```

（3）表存储格式的转换

在前面的创建物理模型章节中，stg_job 表定义为 STORED AS TEXTFILE，是因为导入的爬虫原始数据采用的是文本文件格式，所以 stg_job 表也要采用 TEXTFILE 格式；而 s_job 表定义为 STORED AS SEQUENCEFILE，是因为使用 INSERT OVERWRITE TABLE… SELECT 句式导入数据，可以支持不同存储格式间的自动转换。当然，也可以把 s_job 表定义为 TEXTFILE 格式，源表和目标表采用相同的存储格式。

在创建物理模型时，除了 stg_job 表和 stg_news 表之外，均定义为 SEQUENCEFILE 格式，是因为序列文件采用了压缩格式，可以节省存储空间。

（4）语句运行效率

和 LOAD DATA INTO TABLE 句式一样，INSERT OVERWRITE TABLE…SELECT 句式在运

行时，也是把命令转换成一个 MapReduce 任务来执行。

如果源表中数据行数有百万行，使用存储过程或高级语言编码实现，即使采用多线程方式，运行的效率和可靠性也是很难评估的。而使用 MapReduce 计算框架，每个 DataNode 上都会分配不同数量的 Map 任务，启动多个 JVM 进程来执行工作，分布式环境的优点得以充分体现。

此外，如果 SELECT 模块在运行时（Map 阶段）需要扫描大量的数据进行筛选，而 INSERT 部分（Reduce 阶段）插入的数据量又较小，那岂不是很耗费资源？比如，想把一周内 stg_job 表中的数据，分别插入到 s_job 表的 7 个分区内，就需要对 stg_job 表扫描 7 次。有没有一种高效的方式？

Hive 提供了另一种 INSERT 语法，可以只扫描一次输入数据，然后按照多种方式进行划分。如下例子描述了如何将数据同时插入到 s_job 表的不同分区内。

```
FROM stg_job
INSERT OVERWRITE TABLE s_job
   PARTITION (pt='20150501)
   SELECT * WHERE stg_job.pt='20150501'
INSERT OVERWRITE TABLE s_job
   PARTITION (pt='20150502)
   SELECT * WHERE stg_job.pt='20150502'
INSERT OVERWRITE TABLE s_job
   PARTITION (pt='20150503)
   SELECT * WHERE stg_job.pt='20150503';
```

从 stg_job 表中读取的每条数据都会经过 SELECT…WHERE 进行筛选，多个 SELECT…WHERE 之间是并列关系、独立判断的。源表中的数据经过一次扫描，会写入目标表的多个分区中，或者都不写入。关于这种句式的详细描述，可以参阅《Hive 编程指南》（人民邮电出版社出版）。

（5）关于动态分区

由于 HDFS 写入数据是冗余存储的，创建动态分区是一项消耗资源的操作，在默认情况下动态分区功能是关闭的。Hive 同时要求即使开启动态分区，表中也不能出现所有字段都是分区字段的情况。在${HIVE_HOME}/conf/hive-site.xml 中，与动态分区相关的部分，有 5 个配置项，如表 5.11 所示。

表 5.11　动态分区属性表

属性名称	默认值	含义
hive.exec.dynamic.partition	false	取值 true 表示开启动态分区功能
hive.exec.dynamic.partition.mode	strict	取值 nonstrict 表示允许所有分区都是动态的
hive.exec.max.dynamic.partitions	1000	一条动态分区创建语句可以创建的最大分区数，超出则抛出异常

续表

属性名称	默认值	含义
hive.exec.max.dynamic.partitions.pernode	100	每个 Mapper 或 Reducer 可以创建的最大分区数，超出则抛出异常
hive.exec.max.created.files	100000	允许全局可以创建的最多文件数。Hadoop 计数器会跟踪记录创建文件个数，超出则抛出异常

以上 5 个配置项，可以使用 HQL 命令进行动态配置，示例如下：

```
set hive.exec.dynamic.partition=true
set hive.exec.dynamic.partition.mode=nonstrict
set hive.exec.max.created.files=100000
set hive.exec.max.dynamic.partitions.pernode=100
set hive.exec.max.dynamic.partitions=1000
```

5.6.2 提取维度信息

在数据仓库中，维度表和事实表是密不可分的。事实表是用户/开发人员所要关注的基本内容；维度表则是观察基本内容的角度。比如在“5 月份商品销售量”中，“销售量”就是事实，“5 月份”则是维度。具体到本项目中，事实表 s_job 中存放了大量的原始“业务”数据——招聘职位信息，而检索页面或统计图表中的查询约束条件，可以看作最直观的各种维度分类。

此外，通过提取维度信息，为同一维度下的不同数据取值做汇总，方便在生成报表时提供数据来源，而不用每次都对事实表 s_job 做汇总。

在职位信息查询页面中，有 4 个下拉列表，取值分别如下：

学历要求

显示值	检索值
专科	B1
本科	B2
研究生	B3
其他	B9

工作年限

显示值	检索值
3 年以下	C1
3 至 5 年	C2
5 年及以上	C3
其他	C9

工作地点

显示值	检索值
北京	A1
上海	A2
广州	A3
深圳	A4
其他	A9

月薪范围

显示值	检索值
一至三万	D1
三至五万	D2
五万以上	D3
面议	D8
其他	D9

比如在爬虫抓取的某条招聘信息中，学历要求是“硕士”，薪水待遇是“年薪 30 万起”，这

两个维度信息分别和“研究生（B3）”及“月薪一至三万（D1）”相对应。在 Web 检索页面上，如果“学历要求”和“月薪范围”选择的分别是 B3 值和 D1 值，那么这条招聘信息应该会查出来。

在 5.3.2 节中，我们针对 4 个工作地区、学历要求、工作经验和薪资待遇 4 个检索域，分别建立了 4 个维度表，即 dim_joblocation、dim_edu、dim_workyear 和 dim_salary。这 4 个表的数据来源是 s_job 表，即经过维度清洗后的职位表。数据从 s_job 表流向 4 个维度表，使用 HQL 抽取维度信息的过程，可以用图 5.6 来表示。

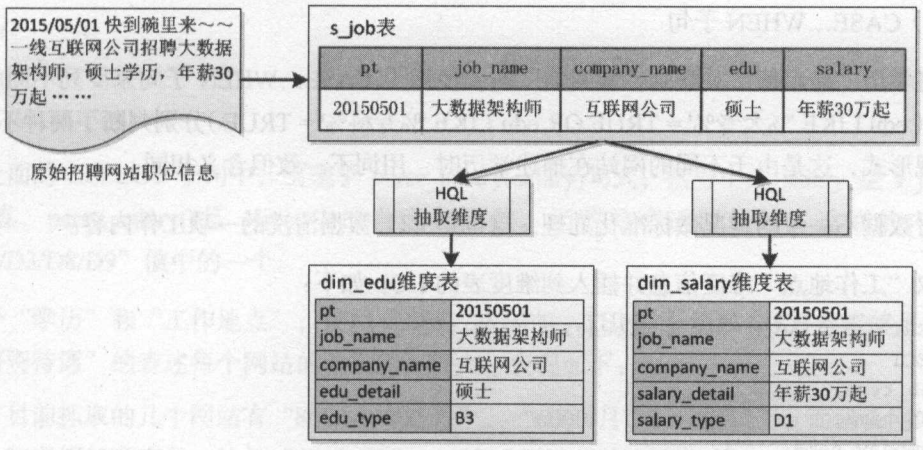


图 5.6　从 s_job 表到维度表的数据流向图

抽取“学历要求”维度信息插入到学历维度表的 HQL 如下：

```
INSERT INTO TABLE dim_edu PARTITION (pt)
SELECT
  web_type,
  job_name,
  company_name,
  edu AS edu_detail,
  CASE
    WHEN (edu LIKE '%大专%' = TRUE OR edu LIKE '%专科%' = TRUE)
    THEN 'B1'
    WHEN (edu LIKE '%本科%' = TRUE)
    THEN 'B2'
    WHEN (edu LIKE '%硕士%' = TRUE OR edu LIKE '%研究生%' = TRUE)
    THEN 'B3'
    ELSE 'B9'
  END
  AS edu_type,
pt
FROM
   s_job
```

```
where
  s_job.pt='20150501';
```

对于上面的 HQL 要注意以下几点。

（1）INSERT INTO TABLE...PARTITION(...) SELECT

和之前的处理方式相同，将维度信息抽取到 dim_edu 表时，采用了创建动态分区的语法，以便可以重复抽取相同分区的维度。

（2）CASE...WHEN 子句

在将学历“显示值”关联到“检索值”时，使用了 CASE...WHEN 子句来罗列不同的学历类型。语法(edu LIKE '%大专%' = TRUE OR edu LIKE '%专科%' = TRUE)分别判断了两种不同的“专科”表现形式，这是由于不同的网站在描述学历时，用词不一致但含义相同。

针对数据不一致的情况做标准化处理，这也是 ETL 数据清洗的一项工作内容。

抽取“工作地点”维度信息并插入到维度表的 HQL 如下：

```
INSERT INTO TABLE  dim_joblocation PARTITION (pt)
SELECT
  web_type,
  job_name,
  company_name,
  job_location AS joblocation_detail,
  CASE
    WHEN (job_location LIKE '%北京%' = TRUE)
    THEN 'A1'
    WHEN (job_location LIKE '%上海%' = TRUE)
    THEN 'A2'
    WHEN (job_location LIKE '%广州%' = TRUE)
THEN 'A3'
    WHEN (job_location LIKE '%深圳%' = TRUE)
    THEN 'A4'
    ELSE 'A9'
  END '
  AS joblocation_type,
  pt
FROM
   s_job
where
  s_job.pt='20150501';
```

该句式和抽取“学历”维度语句相似。WHEN 中的判断条件使用了 LIKE 模糊匹配，而没有使用等值匹配，这是由于有些招聘网站上的工作地点还包含区域信息，比如“北京-海淀区”、“上海徐汇”等。

抽取“工作地点”维度信息的 HQL 如下：

```
INSERT INTO TABLE  dim_salary PARTITION (pt)
SELECT
  web_type,
  job_name,
  company_name,
  salary AS salary_detail,
  parse_salary(salary) AS salary_type,
  pt

FROM
   s_job
WHERE
  s_job.pt='20150501';
```

在上面的 SELECT 子句中，出现了 parse_salary(salary)句式，括号中的 salary 是 s_job.salary 的字段值，parse_salary()是 Hive 中的自定义函数。函数是有返回值的，返回的内容就是“D1/D2/D3/D8/D9”值中的一个。

对于“学历”和“工作地点”，使用 CASE…WHEN…THEN 子句和 LIKE 检索就足以应对；而对“薪资待遇”的表述每个网站的差别就比较大，表现如下：

- 目前抓取的几个网站有“8000-10000/月”、“6000/月”、“面议”，而猎头网站的薪资待遇用年薪表示，比如“30-40 万”、“行业领先薪酬待遇”等。
- dim_salary 维度表中的薪资是以月薪为单位的，那就要把各种文字表现方式都转换成月薪。
- 在 Web 检索页面上，月薪的选择项是几个区间段，如“一至三万”、“三至五万”等，那“年薪 30-40 万”如何归入到区间段中呢？

为了解决以上问题，如果用 HQL 写 CASE…WHEN 子句，复杂度是可想而知的，且代码难以维护，那么有没有一种更好的解决方法呢？

5.7　定义 Hive UDF 封装处理逻辑

使用自定义函数（User Defined Function，UDF）的目的是为了封装 HQL 中整块的处理逻辑，以方便调用。被封装的代码要处理的可能是比较复杂的业务逻辑，如果全部放在 HQL 中，则既不便于阅读，也不便于维护，Hive 支持把这些代码块放在一个函数中，在 HQL 中直接调用得到结果值的做法。

在 RDBMS 中也有自定义函数的概念，比如在 Oracle 中就有函数、过程等数据库对象，方便在数据库开发中重复使用。Oracle 中的函数代码是用 PL/SQL 编写的，创建后便可直接调用；但

在 Hive 自定义函数中出现的通常是 Java 或其他高级语言代码，创建后首先要部署到 Hadoop 分布式环境中才可以在 HQL 中调用。

Hive 安装完成后自带了一些内置函数，使用 SHOW FUNCTIONS 命令可以查看当前 Hive 连接中所有的函数列表，如下列出了该命令运行的部分结果：

```
hive> SHOW FUNCTIONS;
OK
!
!=
%
&
*
+
-
/
<
<=
<=>
<>
=
==
```

如果想使用某个函数，只需要在 HQL 中调用函数名，传入所需的参数列表即可。比如使用 CONCAT 函数实现字符串拼接：

```
0: jdbc:hive://slave02:10001> SELECT  CONCAT('www.','feiguyun','.com') AS website;
+------------------+
|      website     |
+------------------+
| www.feiguyun.com |
+------------------+
1 row selected (31.626 seconds)
0: jdbc:hive://slave02:10001>
```

5.7.1 Hive UDF 的开发、部署和调用

开发 Hive 自定义函数通常使用 Java 语言，这是因为 Hive 本身是用 Java 编写的，用户只需要简单地继承 UDF 类并重载 evaluate 方法即可。

下面描述如何开发 UDF 实现“薪资待遇”自动归类的功能。

① 在 Eclipse 中创建一个 Java 项目。

② 把${hive_HOME}\lib\hive-exec-0.13.0.jar 文件和${HADOOP_HOME}\contrbute\hadoop-common-2.2.0.jar 文件添加到所创建项目的编译路径（build path）中。

③ 创建 ParseSalary 类，继承 org.apache.hadoop.hive.ql.exec.UDF 父类，并重载 evaluate 方法。示例代码如下：

```
package com.feigu.hive.udf;

import org.apache.hadoop.hive.ql.exec.UDF;
import org.apache.hadoop.io.Text;

/**
 * 转换薪资范围
 */
public class ParseSalary extends UDF {

    // 定义工作年限范围
    private static String SALARY_D1 = "D1"; // 一至三万
    private static String SALARY_D2 = "D2"; // 三至五万
    private static String SALARY_D3 = "D3"; // 五万以上
    private static String SALARY_D8 = "D8"; // 面议
    private static String SALARY_D9 = "D9"; // 其他

    public Text evaluate(final Text s) {
        if (s == null) {
            return new Text(SALARY_D9);
        }
        return new Text(parseSalary(s.toString()));
    }

    private String parseSalary(String word) {
        try {
          float money = 0;
         if (word.indexOf("-") > -1 && word.endsWith("万")) {
            word = word.replace("万", "");
            String[] ary = word.split("-");
            // 如果结束薪资小于 5 万，则说明是月薪，否则是年薪
            if (Integer.parseInt(ary[0]) <= 5) {
                money = (Float.parseFloat(ary[0]) + Float.parseFloat(ary[1])) / 2;
            } else {
                money = (Float.parseFloat(ary[0]) + Float.parseFloat(ary[1])) / 24;
            }
            return parseSalary2(money);
        } else {
            if (word.indexOf("-") > -1 && word.endsWith("/月")) {
                word = word.replace("/月", "");
                String[] ary = word.split("-");
                //处理月薪 15000-30000/月的格式
                money = (Float.parseFloat(ary[0]) + Float.parseFloat(ary[1])) / 20000;
                  return parseSalary2(money);
              }
```

```
                else {
                  if (word.endsWith("万")) {
                     money = Float.parseFloat(word.replaceAll("万", ""));
                     return parseSalary2(money);
                  } else {
                     if (word.indexOf("面议") > -1) {
                        return SALARY_D8;
                     } else {
                        return SALARY_D9;
                     }
                  }
                }
             }
          } catch (Exception e) {
             return SALARY_D9;
          }
       }
```

代码中的“public Text evaluate(final Text s)”是自定义函数执行的入口方法，输入参数 Text s 是转换前的 salary 值，返回值是 Text 类型的字符串。

④ 代码编写完成后，使用 Ant 打包成 JAR 包，命名为“feigu_hiveUDF.jar”。

⑤ 目前该 JAR 包还在开发机器上，要使用 hadoop fs 命令上传到 HDFS 中，以便分布式环境中的其他计算节点（Task Tracker）也能引用到。

```
hadoop fs -copyFromLocal feigu_hiveUDF.jar hdfs://master:49100/user/hadoop/feigu_hiveUDF.jar
```

```
feigu@slave02:~$ hadoop fs -ls  hdfs://master:49100/user/hadoop/feigu_hiveUDF.jar
-rw-r--r--   2 feigu supergroup       4988 2015-08-10 17:03 hdfs://master:49100/user/hadoop/feigu_hiveUDF.jar
```

⑥ 把 HDSF 中的 JAR 包引入到 Hive 运行环境中，使用 ADD JAR 命令实现。

```
hive> ADD JAR hdfs://master:49100/user/hadoop/feigu_hiveUDF.jar;
converting to local hdfs://master:49100/user/hadoop/feigu_hiveUDF.jar
Added /tmp/7c3c3cbd-0bf3-40c4-8428-18fb23670d45_resources/feigu_hiveUDF.jar to class path
Added resource: /tmp/7c3c3cbd-0bf3-40c4-8428-18fb23670d45_resources/feigu_hiveUDF.jar
```

⑦ 在 Hive 运行环境中创建自定义函数 parse_salary，并指定它的实现方法是 com.feigu.hive.udf.ParseSalary。ParseSalary 是刚才在 Java 项目中创建的类名，类名前是包名。

```
hive> CREATE TEMPORARY FUNCTION parse_salary AS 'com.feigu.hive.udf.ParseSalary';
OK
Time taken: 0.055 seconds
```

⑧ 最后，在 HQL 中就可以直接使用 parse_salary 函数了。

```
0: jdbc:hive://slave02:10001> SELECT
. . . . . . . . . . . . . . >   salary AS salary_detail,
. . . . . . . . . . . . . . >   parse_salary(salary) AS salary_type
. . . . . . . . . . . . . . > FROM
. . . . . . . . . . . . . . >   s_job
. . . . . . . . . . . . . . > WHERE
. . . . . . . . . . . . . . >   pt='20150501';
+----------------+--------------+
| salary_detail  | salary_type  |
+----------------+--------------+
| 20-40万        | D1           |
| 20-30万        | D1           |
| 20-30万        | D1           |
| 面议           | D8           |
| 面议           | D8           |
| 面议           | D8           |
| 面议           | D8           |
| 面议           | D8           |
| 面议           | D8           |
| 面议           | D8           |
| 面议           | D8           |
| 面议           | D8           |
| 面议           | D8           |
| 面议           | D8           |
| 面议           | D8           |
| 面议           | D8           |
| 面议           | D8           |
+----------------+--------------+
17 rows selected (16.619 seconds)
```

通过以上几个步骤，实现了 Hive UDF 的简单开发和使用。有关 Hive UDF 的更多用法，比如如何同时接收多个输入参数，如何让 UDF 返回一个结果集，可以参考《Hive 编程指南》（人民邮电出版社出版）第 13 章的内容。

5.7.2　Python 版本的 UDF

倘若不熟悉 Java 语言，是不是就无法开发 Hive 自定义函数了呢？

利用 Hadoop Streaming 技术可以解决这个问题。

Hadoop 的计算框架 MapReduce 本身是用 Java 语言开发的，用户要想在这个计算框架上实现自己的应用，就必须使用 Java 开发相应的代码，这无疑使得操作者的学习曲线变陡了。Hadoop 在设计之初就考虑到了这一点，使用 Streaming 技术来替代 Java 编程，允许用户使用其他语言实现业务逻辑处理。

Streaming 采用 UNIX 标准输入输出机制（stdin/stdout）作为应用程序和 Hadoop 计算框架之间的数据接口标准，这样就很好地实现了语言无关性，只要符合标准 I/O 接口，开发人员便可以选择任意语言编写 Map/Reduce 模块。Streaming 模式的流程，如图 5.7 所示。

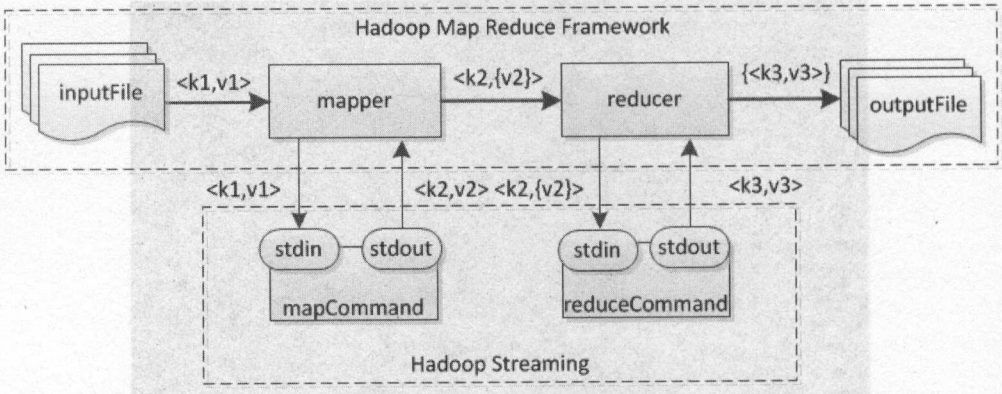


图 5.7　Streaming 模式流程图

使用非 Java 语言开发 Hive UDF 的步骤和上一节类似，也分为开发、部署、调用几个步骤。

下面演示如何使用 Python 语言实现月薪范围的归类。parseSalary.py 代码如下：

```
# encoding: utf-8
#!/usr/bin/python
import sys
SALARY_D1 = 'D1' #一至三万
SALARY_D2 = 'D2' #三至五万
SALARY_D3 = 'D3' #五万以上
SALARY_D8 = 'D8' #面议
SALARY_D9 = 'D9' #其他
def parseSalary(word):
    try:
        if (word.find('-')>-1 and word.endswith('万')):
            word = word.replace('万','')
            ary = word.split('-')
            st = ary[0]
            ed = ary[1]
            '''如果结束薪资小于 5 万，则说明是月薪，否则是年薪
            '''
            if (int(ed)>=5):
                money = (int(st)+int(ed))/2
            else:
                money = (int(st)+int(ed))/24
            return parseSalary2(money)
        elif (word.endswith('万')):
            money = word.replace('万','')
            return parseSalary2(money)
        elif (word.find('面议')>-1):
            return SALARY_D8
        else:
```

```
            return SALARY_D9
        except Exception as e:
            print e
            return SALARY_D9

def parseSalary2(money):
    try:
        if (money<3):
            return SALARY_D1
        if (money>=3 and money<5):
            return SALARY_D2
        if (money>=5):
            return SALARY_D3
    except Exception as e:
        print e
        return SALARY_D9

for line in sys.stdin:
    line = line.strip()
    print parseSalary(line)
```

上面代码中的“for line in sys.stdin”位置是自定义函数的入口，sys.stdin 表示从标准输入管道中得到输入内容，即 stg_job 表的 salary 字段的值，“print parseSalary(line)”是把处理后的结果打印到标准输出，也就是返回给 HQL 的调用处。

Python 代码开发完成后，可以直接在命令行中测试，测试方法如下：

```
$ python  文件名.py  <回车>
```

如下测试了所抓取的几种薪资内容，均返回对应的月薪范围：

```
feigu@slave02:~$ python parseSalary.py
20-30万
D1
20-40万
D1
面议
D8
```

注意：上述代码在运行时，从控制台接收输入，输入“20-30 万”后按回车键，然后按“Ctrl+D”快捷键提交输入，“D1”是返回的值，也打印到控制台上。具体的运行方法可以参考 Python 相关语法手册。

部署和调用的方式如下。首先把 Python 文件上传到 HDFS 文件系统：

```
feigu@slave02:~$ hadoop fs -copyFromLocal parseSalary.py  hdfs://master:49100/user/hadoop/parseSalary.py
feigu@slave02:~$ hadoop fs -ls  hdfs://master:49100/user/hadoop/parseSalary.py
-rw-r--r--   2 feigu supergroup       1757 2015-08-10 17:44 hdfs://master:49100/user/hadoop/parseSalary.py
```

然后，在 Hive 中使用 ADD FILE 命令加载该 Python 文件：

```
ADD FILE hdfs://master:49100/user/hadoop/parseSalary.py;
```

最后，在 HQL 中使用 TRANSFORM 函数动态执行 Python 文件，得到结果：

```
0: jdbc:hive://slave02:10001> SELECT
. . . . . . . . . . . . . . . >   TRANSFORM (salary)
. . . . . . . . . . . . . . . >    USING 'python parseSalary.py'
. . . . . . . . . . . . . . . >    AS salary_type
. . . . . . . . . . . . . . . >  FROM
. . . . . . . . . . . . . . . >    s_job
. . . . . . . . . . . . . . . >  WHERE
. . . . . . . . . . . . . . . >    pt='20150501';
+--------------+
| salary_type  |
+--------------+
| D1           |
| D1           |
| D1           |
| D8           |
| D8           |
| D8           |
| D8           |
| D8           |
| D8           |
| D8           |
| D8           |
| D8           |
| D8           |
| D8           |
| D8           |
| D8           |
| D8           |
+--------------+
17 rows selected (15.218 seconds)
```

使用 Streaming 技术虽然实现了语言无关性，但也会有一些限制：

- 通常 Streaming 的执行效率会比对应的编写 UDF 的方式要低。管道中序列化/反序列化数据通常是低效的。
- 采用 Streaming 技术后，由于使用了不同的编程语言，使得程序调试变得困难。

如果采用脚本语言编写 UDF，比如 PHP/Python/Ruby，则必须保证每个计算节点（Task Tracker）上都已经安装了支持脚本运行的环境，因为 Job 是分发到所有节点上运行的。

5.8 使用左外连接构造聚合表 rpt_job

虽然事实表中的数据很详细，但是在需要进行汇总、均值、分组等处理后再显示报表的情况下，大量的数据在内存中运算会消耗很多性能。这时就可以对一些底层事实表预先进行处理，比

如预先进行汇总、按条件分组，形成一个新的结果集，用户检索是基于这个数据量少的结果集的。这个结果集被称作聚合表，形成聚合表的过程叫作聚合过程。

在 SQL 理论中，JOIN（连接）是通过一些条件关联两个集合，连接两个集合中的元素，得到一个新的结果集的操作。Hive 支持通常的 SQL JOIN 语句，但是只支持等值连接。连接又可以分为内连接（INNER JOIN）和外连接（OUTER JOIN）。在内连接情况下，只有进行连接的两个表中都存在与连接标准相匹配的数据才会被展示出来；而在外连接情况下，两个表存在主附关系，能够保证主表中的数据全部展示出来，附表中的数据若匹配则展示出来，否则返回 NULL。

左外连接通过关键字 LEFT OUTER 进行标识。在这种连接操作中，JOIN 操作符左侧表中符合 WHERE 子句的所有记录会被返回；JOIN 操作符右侧表中如果没有符合 ON 后面的连接条件记录时，右侧表返回的列值是 NULL。

在 5.3.2 节中，最后设计了一个 rpt_job 表，其中的数据是要导入到 HBase 表中的。4 个维度表的信息需要和 rpt_job 表进行关联，从而得到聚合表的完整数据。左外连接同样使用 HQL 完成，把 s_job 表的数据作为主表（左侧表），4 个维度表作为右侧表，维度表按照日期分区、职位名、公司名和维度名 4 个字段值和 s_job 表对应的字段做等值连接，得到一个既包含 s_job 表的所有字段，又包含 4 个维度表的维度类型的聚合表。s_job 表左外连接的过程如图 5.8 所示。

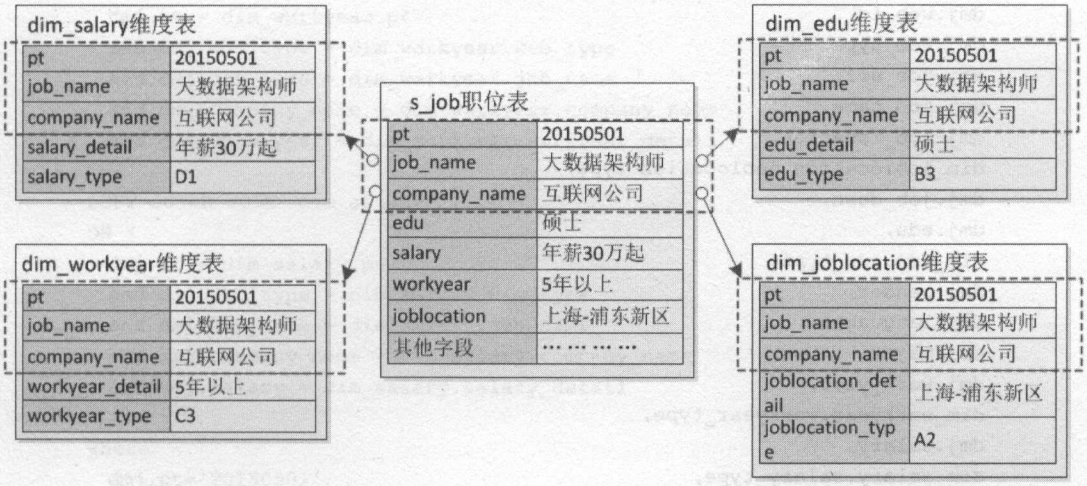


图 5.8　s_job 表左外连接的过程

左外连接后生成的 rpt_job 表中的数据，如图 5.9 所示。

rpt_job职位表

pt	20150501
job_name	大数据架构师
company_name	互联网公司
edu_detail	硕士
edu_type	B3
salary_detail	年薪30万起
salary_type	D1
workyear_detail	5年以上
workyear_type	C3
joblocation_detail	上海-浦东新区
joblocation_type	A2
其他字段	…… …… …… ……

图 5.9　左外连接后生成的 rpt_job 表中的数据

左外连接所使用的 HQL 如下：

```
INSERT INTO TABLE
      rpt_job
   PARTITION (pt)
   SELECT
     dmj.web_id,
     dmj.web_type,
     dmj.job_url,
     dmj.job_name,
     dmj.job_location,
     dim_joblocation.joblocation_type,
     dmj.job_desc,
     dmj.edu,
     dim_edu.edu_type,
     dmj.gender,
     dmj.language,
     dmj.major,
     dmj.work_year,
     dim_workyear.workyear_type,
     dmj.salary,
     dim_salary.salary_type,
     dmj.company_name,
     dmj.company_desc,
     dmj.company_address,
     dmj.company_worktype,
     dmj.company_scale,
     dmj.company_prop,
```

```
  dmj.company_website,
  dmj.curl_timestamp,
  dmj.vip_flg,
  dmj.pt
FROM
  dm_job dmj
LEFT OUTER JOIN  dim_joblocation
ON (
  dmj.pt = dim_joblocation.pt
  and dmj.web_type = dim_joblocation.web_type
  and dmj.job_name = dim_joblocation.job_name
  and dmj.company_name = dim_joblocation.company_name
  and dmj.job_location = dim_joblocation.joblocation_detail
 )
LEFT OUTER JOIN  dim_edu
ON (
  dmj.pt = dim_edu.pt
  and dmj.web_type = dim_edu.web_type
  and dmj.job_name = dim_edu.job_name
  and dmj.company_name = dim_edu.company_name
  and dmj.edu = dim_edu.edu_detail
 )
 LEFT OUTER JOIN  dim_workyear
ON (
  dmj.pt = dim_workyear.pt
  and dmj.web_type = dim_workyear.web_type
  and dmj.job_name = dim_workyear.job_name
  and dmj.company_name = dim_workyear.company_name
  and dmj.work_year = dim_workyear.workyear_detail
 )
LEFT OUTER JOIN  dim_salary
ON (
  dmj.pt = dim_salary.pt
  and dmj.web_type = dim_salary.web_type
  and dmj.job_name = dim_salary.job_name
  and dmj.company_name = dim_salary.company_name
  and dmj.salary = dim_salary.salary_detail
 )
 where
  dmj.pt='20150501';
```

s_job 表分别和 dim_joblocation、dim_edu、dim_workyear 和 dim_salary 表依次进行连接。在书写 HQL 时，4 个维度表出现的先后次序没有特别的限制，但是 Hive 在执行时是按照从左到右的顺序执行的，即最先和 dim_joblocation 表进行关联，生成的结果集再和 dim_edu 表关联，依此类推。

5.9 让数据处理自动调度

前面我们讨论了如何利用 HQL 实现数据加载、清洗、转换，ETL 的每一个步骤最终都会表现为一个或一组 HQL 语句。对于一个数据来源复杂、处理逻辑烦琐的生产场景而言，每一次的 ETL 过程都会用到很多 HQL，并且这些 HQL 之间的数据有依赖关系，必须按照先后次序运行。

比如本项目中的数据来源，有各种不同的招聘网站。如果要汇总分析的是各大电商网站的商品信息，那么数据来源和每天的增量数据都会非常多。此外，dim_edu 维度表的数据来源于 s_job 表；s_job 表的数据又是从 stg_job 表清洗之后得到的；而 stg_job 表的数据来源是从文件导入的，表数据之间也存在依赖关系，HQL 的运行次序一定要确保正确。那么，如何来保证 ETL 中的所有 HQL 一个不漏地顺序执行呢？

上面章节在演示时，都是把每个 HQL 放在 Hive CLI 界面中单独运行的，对于创建物理模型 DDL 来讲，只运行一次也就够了。但爬虫数据是每天都要增量获取的，ETL 部分的 HQL 是每天都要运行的，只有通过程序自动调度，才是值得信赖的解决办法。

5.9.1 HQL 的几种执行方式

在前面章节中展示过两种 ETL 运行方式，分别是使用 Hive CLI（Command Line Interface，命令行接口）和 beeline 客户端。安装 Hive 后，在 bin 目录下有两个可执行脚本，分别是小写的 hive 和 beeline 文件名，如下所示：

```
feigu@slave02:/home/hadoop/bigdata/hive/bin$ ls -l
total 36
-rwxr-xr-x 1 hadoop hadoop  881 May 24 16:11 beeline
drwxrwxr-x 3 hadoop hadoop 4096 May 24 16:11 ext
-rwxr-xr-x 1 hadoop hadoop 7164 May 24 16:11 hive
-rwxr-xr-x 1 hadoop hadoop 1900 May 24 16:11 hive-config.sh
-rwxr-xr-x 1 hadoop hadoop  885 May 24 16:11 hiveserver2
-rw-rw-r-- 1 hadoop hadoop   28 Jul  7 22:42 hiveserver-nohup.out
-rwxr-xr-x 1 hadoop hadoop  832 May 24 16:11 metatool
-rwxr-xr-x 1 hadoop hadoop  884 May 24 16:11 schematool
```

Hive 命令行界面，也就是 CLI，是和 Hive 交互的最常见方式。使用 CLI，用户可以创建表、检查模式以及运行 HQL 语句等。

使用如下命令可以查看 CLI 所提供的选项列表：

```
feigu@slave02:/home/hadoop/bigdata/hive/bin$ hive --cli --help
Unrecognized option: --cli
usage: hive
 -d,--define <key=value>          Variable subsitution to apply to hive
                                  commands. e.g. -d A=B or --define A=B
    --database <databasename>     Specify the database to use
 -e <quoted-query-string>         SQL from command line
 -f <filename>                    SQL from files
 -h <hostname>                    connecting to Hive Server on remote host
 -H,--help                        Print help information
    --hiveconf <property=value>   Use value for given property
    --hivevar <key=value>         Variable subsitution to apply to hive
                                  commands. e.g. --hivevar A=B
 -i <filename>                    Initialization SQL file
 -p <port>                        connecting to Hive Server on port number
 -S,--silent                      Silent mode in interactive shell
 -v,--verbose                     Verbose mode (echo executed SQL to the
                                  console)
```

beeline 工具是 Hive 0.11.0 版本中增加的新的交互式 CLI，它基于 SQLLine，可以作为 Hive JDBC 客户端访问 Hive 中的表和数据，执行 HQL 语句。运行 beeline 的前提是要先启动 HiveServer 服务。beeline 在使用前要先连接上 HiveServer 服务，有点像 PL/SQL 客户端工具的角色，如下所示：

```
feigu@slave02:/home/hadoop/bigdata/hive/bin$ ./beeline
Beeline version 0.13.1 by Apache Hive
beeline> !connect jdbc:hive://slave02:10001
scan complete in 3ms
Connecting to jdbc:hive://slave02:10001
Enter username for jdbc:hive://slave02:10001: hive
Enter password for jdbc:hive://slave02:10001: ****
Connected to: Hive (version 1)
Driver: Hive JDBC (version 0.13.1)
```

使用“!connect jdbc:hive://hiveServerIP 地址:端口号”的格式连接 HiveServer。目前 Hive 提供两种 JDBC 连接方式，分别是 HiveServer 和 HiveServer2，两者的区别下面会讲到。

beeline 在连接时要识别用户身份，可以通过配置 Kerberos 认证来实现；而 Hive CLI 连接时是不需要用户登录的。HQL 在 beeline 中均可以正常使用，和 Hive CLI 的区别是不再输出 MapReduce 任务执行的日志信息，并且连接 HiveServer 后，所有的 HQL 语句执行均使用同一个数据库连接。

下面展示了 HQL 运行内容：

```
0: jdbc:hive://slave02:10001> SHOW DATABASES;
+----------------+
| database_name  |
+----------------+
| default        |
| demo           |
| feigu3         |
| test           |
+----------------+
4 rows selected (0.015 seconds)
0: jdbc:hive://slave02:10001> SHOW PARTITIONS dim_resume_techword;
+--------------+
|  partition   |
+--------------+
| pt=20150624  |
| pt=20150803  |
| pt=20150804  |
| pt=20150805  |
| pt=20150806  |
| pt=20150807  |
| pt=20150808  |
| pt=20150809  |
| pt=20150810  |
+--------------+
9 rows selected (0.046 seconds)
```

```
0: jdbc:hive://slave02:10001> SELECT COUNT(*) AS CNT FROM s_job WHERE pt='20150501';
+------+
| cnt  |
+------+
| 17   |
+------+
1 row selected (22.043 seconds)
```

退出 Hive CLI 使用 exit 命令，退出 beeline 使用!q 命令。

另外，还有一种在生产场景中比较实用的运行方式，是把 HQL 嵌入 shell 脚本中执行。下面是一个简单的例子（printJobName.sh）。

```
#!/bin/sh
#######
#   从控制台接收日期参数
#   打印该日的职位名称
#######
function printJobNameByDate() {
    jobdate=$1
    echo "USE feigu3; SELECT  job_name FROM s_job WHERE pt='$jobdate';" | hive
}
printJobNameByDate $1
```

执行方式是在控制台输入“sh printJobName.sh 20150501<回车>”，即可在 shell 脚本中运行 HQL 并打印结果。

上面的代码是把参数和 HQL 拼接成字符串，打印到管道“|”中，hive 命令从管道中取得命令行，得以执行。在 echo 后可以拼接多条 HQL 语句，中间用分号分隔。使用这种方式要注意的是：HQL 不宜太复杂；输出到控制台的内容混合了数据和日志信息；在运行脚本前，还要保证 hive 命令已经加入到系统 PATH 环境变量中。

5.9.2　Hive Thrift 服务

在使用 beeline 连接 Hive 数据仓库前，要在服务器上开启 Hive Thrift 服务。Hive Thrift（HiveServer）服务是 Hive 中的组件之一，设计目的是为了实现跨语言轻量级访问 Hive 数据仓库。

通过 Hive CLI 访问 Hive 是胖客户端方式，需要客户机安装 JRE 环境和 Hive 程序，就像客户端要连接 Oracle 数据库服务器，必须安装 Oracle 客户端一样。而采用 JDBC 方式访问 Hive 数据仓库，则是瘦客户端方式，前提是服务器必须先启动 Hive Thrift 服务。和使用 JDBC 方式访问 Oracle 数据库，要求服务器要启动 LISTENER 监听服务，道理是一样的。

目前，HiveThrift 组件包含两个版本，分别是 HiveServer（ThriftServer）和 HiveServer2，后者是在 Hive 0.11.0 版本中增加的，原因是 HiveServer 不能处理多客户端的并发请求。需要注意的是，HiveServer2 不会向下兼容 ThriftServer，编码实现时必须分别处理。两者的差别有三：

（1）启动方式

HiveServer	hive --service hiveserver -p <端口号>
HiveServer2	hive --service hiveserver2 -hiveconf hive.server2.thrift.port=<端口号> 或者，在${HIVE_HOME}\bin\下运行 HiveServer2 脚本，端口等配置信息默认从 hive-site.xml 中读取

（2）beeline/JDBC 连接字符串

HiveServer	jdbc:hive://hive 服务器 IP 地址:端口号
HiveServer2	jdbc:hive2://hive 服务器 IP 地址:端口号

（3）JDBC 驱动名称

HiveServer	org.apache.hadoop.hive.service.HiveServer
HiveServer2	org.apache.hive.service.server.HiveServer2

需要注意的是，两个组件启动后都是独立的 Java 进程，只要端口号不重复，在一台机器上就可以同时启动两个服务。此外，HiveService 服务可以选择在 Hadoop 集群环境中的任何一个安装

有 Hive 的节点上启动，不必一定要放在 NameNode 上。

5.9.3 使用 JDBC 连接 Hive

使用 JDBC 接口操作 Hive 数据库，和操作其他数据库的方式大体相同，步骤分为查找驱动程序、得到数据源连接、创建 Statement/PreparedStatement 语句、执行 HQL、关闭数据源连接。下面代码演示了从 daily_dim_sum 表读取某个分区中数据的功能。

```
public void prepStatement() throws Exception, SQLException{
// 加载 HiveServer 驱动
Class.forName("org.apache.hadoop.hive.jdbc.HiveDriver");
String url = "jdbc:hive://slave02:10001/default";
    Connection dbConn =DriverManager.getConnection(url);
    // 使用预编译 Statement 执行 HQL
    String hql = "SELECT dim_type, cnt_val FROM feigu3.daily_dim_sum WHERE pt=?";
    PreparedStatement pstmt = dbConn.prepareStatement(hql);
    pstmt.setString(1, "20150501");
    ResultSet rs = pstmt.executeQuery();
    // 循环结果集
    while (rs.next()) {
        System.out.println(rs.getString(1) + "," + rs.getString(2));
    }
    // 关闭连接
    rs.close();
    dbConn.close();
}
```

需要注意的是，5.6 节中讨论的用于数据清洗转换的 ETL HQL，都是“LOAD DATA”、“SELECT...OVERWRITE”数据写入类的，不是查询类的。在 JDBC 代码中，要使用 Statement 的 executeUpdate()方法，而不是 executeQuery()方法。

另外，由于要执行的 HQL 语句较多，为了便于维护查看，可以把这些 HQL 单独放入资源文件中，在 JDBC 代码中依次调用执行。示例如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<feigu3HQL>
    <hql name="set1">
        <![CDATA[
        set hive.exec.dynamic.partition=true
        ]]>
    </hql>
    <hql name="drop_stgjob">
        <![CDATA[
        ALTER TABLE stg_job DROP PARTITION (pt='$DATE$')
        ]]>
</hql>
```

```
    <hql name="load_stgjob">
        <![CDATA[
        load data local inpath '$DIR$' overwrite into table stg_job partition
(pt='$DATE$')
        ]]>
    </hql>
    <hql name="drop_dimedu">
        <![CDATA[
         ALTER TABLE  dim_edu DROP PARTITION (pt='$DATE$')
        ]]>
    </hql>
    <hql name="ins_dimedu">
        <![CDATA[
        INSERT into TABLE
           dim_edu
        PARTITION (pt)
```

1. 封装 HQL 的 XML 文件

代码片段如下：

```
        SELECT
          web_type,
          job_name,
          company_name,
          edu AS edu_detail,
          CASE
            WHEN (edu LIKE '%大专%' = TRUE OR edu LIKE '%专科%' = TRUE)
            THEN 'B1'
            WHEN (edu LIKE '%本科%' = TRUE)
            THEN 'B2'
            WHEN (edu LIKE '%硕士%' = TRUE OR edu LIKE '%研究生%' = TRUE)
            THEN 'B3'
            ELSE 'B9'
          END
          AS edu_type,
          pt
        FROM
           s_job
        where
          s_job.pt='$DATE$'
        ]]>
</hql>
</feigu3HQL>
```

2. 在 Java 代码中依次调用 HQL 执行

代码片段如下：

```
logger.info("连接 Hive Thrift Server 开始.....");
if (!dbTool.connHive()){
```

```
        logger.error("连接 Hive Thrift Server 失败");
        return;
    }
    HQLTool.initHQLMap();
    logger.info("设定 Hive 支持动态增加分区");
    dbTool.ExecHQL(HQLTool.getHQL("set1"));
    dbTool.ExecHQL(HQLTool.getHQL("set2"));

    logger.info("切换数据库");
    String sql = HQLTool.getHQL("use_db").replace("$DBNAME$", hiveDB);
    dbTool.ExecHQL(sql);

    logger.info("删除 sjob 分区{}开始.",date);
    sql = HQLTool.getHQL("drop_stgjob").replace("$DATE$", date);
    dbTool.ExecHQL(sql);
    jobdata_dir = jobdata_dir + date + "/";
    logger.info("导入{}目录下的文件到 stg_job 开始...",jobdata_dir);
    sql = HQLTool.getHQL("load_stgjob").replace("$DIR$", jobdata_dir);
    sql = sql.replace("$DATE$", date);
    dbTool.ExecHQL(sql);
    logger.info("删除分区结束.");
    //依次执行 XML 文件中的 HQL
    ......
```

开发 Java JDBC 程序，数据库驱动 JAR 包是必不可少的，通常都由数据库厂商提供，Hive 也不例外。在${HIVE_HOME}\lib 目录下，有 4 个 JAR 包：hive-jdbc.jar、hive-exec.jar、hive_metastore、hive_service.jar，在开发阶段都要加入项目的 CLASSPATH 类路径中。此外，hive 命令的执行又依赖于 Hadoop 环境，所以在实际运行环境中，又要把 Hadoop 依赖 JAR 包输出到环境变量中。

通常 Hive 程序开发是在 Windows 的 Eclipse 等 IDE 环境中完成的，而测试/生产环境又都是 Linux 平台。由于 MapReduce 任务是分布在多台机器上运行的，使用 IDE 工具无法进行断点调试，遭遇运行期错误时常常会感到无从下手。下面对 Hive 调试常见问题做一个总结。

（1）HQL 及 Java 代码

此类问题初学者可能会遇到，Java 编码方面的语法问题可以用 IDE 工具解决，HQL 运行出错多半是语法问题。可以先把 HQL 在 beeline 客户端里面运行一下，保证语法正确了再放入代码中。HQL 语法是不区分大小写的，但是规范的写法是保留字全部使用大写。

（2）Hive 环境

这类错误都是 SQLException 类型的，比如 java.sql.SQLException: Query returned non-zero code: 2, cause: FAILED: Execution Error, return code 2 from org.apache.hadoop.hive.ql.exec.MapRedTask 这种类型的错误，JDBC 直接抛出 Hive 异常，从异常信息中又读不出错误原因。毫无疑问，遇到这

种问题时要查看日志信息。Hive 中的日志分为两种，如表 5.12 所示。

表 5.12　Hive 中的两种日志

	系统日志	Job 日志
用途	记录了 Hive 的运行情况、错误状况	记录了 Hive 中 Job 执行的历史过程

存储位置	采用 Log4j 日志方式
	在 hive/conf/hive-log4j.properties 文件中记录了 Hive 日志的存储位置，默认配置项如下：
	hive.root.logger=WARN,DRFA
	hive.log.dir=${java.io.tmpdir}/${user.name} # 默认的存储位置
	hive.log.file=hive.log　# 默认的文件名
		采用 Log4j 日志方式
		在 hive/conf/hive-sites.xml 中有 hive.querylog.location 属性，比如：
		<property>
		<name>hive.querylog.location</name>
		<value>/home/hadoop/opt/bigdata/logs/hive/logs</value>
		<description>
		Location of Hive run time structured log file
		</description>
		</property>

（2）Hadoop 环境相关

由于 Hive HQL 语句运行基本上都要转换成 MapReduce 任务，因此直接查看 Hadoop Job 运行日志也是最快捷的方法。在大多数应用场景中，直接查看生产环境的运行日志调错是不允许的，可以通过 Hadoop 提供的 Web 监控界面来实现。在浏览器中访问集群 master 节点的默认 9005 端口可以查看所有任务的运行状态和详细信息，当然，其中也会包含 Hive 提交的 HQL 的运行情况，如图 5.10 所示。

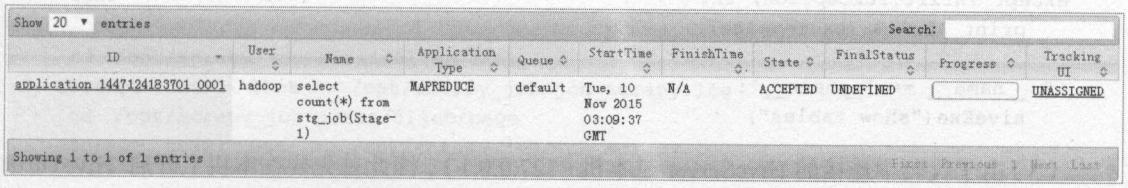


图 5.10　查看任务的运行状态和详细信息

此外，如果已经安装了 CDH HUE 或 Zeus 等集群任务管理工具，在其提供的界面中查看日志信息也非常方便。

5.9.4　Python 调用 HiveServer 服务

使用 Java JDBC 方式操作 Hive 表中的数据，java 文件要编译成 class 文件才可以使用，并且还需要 Hive 的 JAR 包支持。HiveThriftSever 也支持用其他语言连接的方式，比如 Python 语法简单，并且以脚本方式运行，免去了编译的环节。

下面例子展示了如何使用 Python 调用 HiveServer。

（1）找到${HIVE_HOME}/lib/py 目录，该目录下的文件是 Hive 提供的使用 Python 语言连接 HiveServer 的第三方类库，都是以.py 结尾的 Python 文件。

（2）在 py 目录下创建 showHiveTable.py 文件，文件代码如下：

```
import sys
from hive_service import ThriftHive
from hive_service.ttypes import HiveServerException
from thrift import Thrift
from thrift.transport import TSocket
from thrift.transport import TTransport
from thrift.protocol import TBinaryProtocol

def hiveExe(sql):
try:
    transport = TSocket.TSocket('127.0.0.1', 10000)
    transport = TTransport.TBufferedTransport(transport)
    protocol = TBinaryProtocol.TBinaryProtocol(transport)
    client = ThriftHive.Client(protocol)
    sport.open()
    client.execute('use feigu3')
    client.execute(sql)

    print "Tables in feigu3 : "
    print client.fetchAll()
    transport.close()
except Thrift.TException, tx:
    print '%s' % (tx.message)

if __name__ == '__main__':
    hiveExe("show tables")
```

（3）由于代码中连接的 HiveServer 是本机（127.0.0.1），因此首先要在本机上开启 HiveThrift 服务，然后在控制台上直接运行该文件。

```
python  showHiveTable.py
```

如果运行正常，则可以看到 feigu3 数据库中所有的表名。

Python 和 HiveServer 通信使用 Socket 方式，代码中以“from”开头的内容都是引用 Hive 库函数，这些库函数类和 showHiveTable.py 文件位于同级目录下，以便能被该文件引用到。也可以把 py 目录下的东西加入到 Python 的第三方包路径中，这样 showHiveTable.py 就可以在任何一个目录位置运行了。有关 Python 包路径的设置和引用，读者可以参阅 Python 书籍。

5.9.5　用 crontab 实现的任务调度

至此，我们已经详细介绍了如何用爬虫抓取数据，以及最后落地到 Hive 数据仓库中。虽然爬虫实现了根据日期抓取招聘信息功能，但是何时运行爬虫程序，是代码本身无法解决的。启动爬虫的操作和进行 Hive ETL 数据清洗的流程，是由操作人员手工触发的。在实际生产环境中，通常人工操作只会出现在系统异常时必须手动干预的条件下。一般而言，程序都是通过任务调度的方式自动执行的，当执行出错无法继续时，会发出报警信息，以短信或邮件方式通知运维人员。

一个可靠的调度系统对于整个系统的稳健运行是至关重要的。调度系统的实现方式也有多种，可以由公司自行研发，或者采购现有的成熟产品。操作系统本身也提供了任务调度的功能，比如 Windows 系统的“计划任务”和 Linux 系统的 crontab 等。

这里需要实现任务调度的功能有两个，分别是爬虫的自动抓取和 Hive ETL 任务的自动执行，均使用 Linux 的 crontab 功能实现。crontab 可以让 Linux 在某个时间点自动执行一段命令行语句，基本用法可以参阅 Linux 文档。

Python 爬虫的执行是通过在终端输入命令行“scrapy crawl <project_name>”完成的。我们可以把启动每个爬虫的命令行都放在一个 shell script 文件中，在 crontab 中调用该脚本，让爬虫自动运行。shell 脚本示例代码如下：

```
#!/bin/sh
PATH=$PATH:/usr/local/bin
export PATH
date
echo "抓取 job 开始>>>"

cd  /opt/scrapy_job/sites/51job/link
scrapy  crawl  link >> /opt/scrapy_job/job_scrapy.log
cd  /opt/scrapy_job/sites/51job/page
scrapy  crawl  page >> /opt/scrapy_job/job_scrapy.log

cd  /opt/scrapy_job/sites/liepin/link
scrapy crawl link >> /opt/scrapy_job/job_scrapy.log
cd  /opt/scrapy_job/sites/liepin/page
scrapy crawl page >> /opt/scrapy_job/job_scrapy.log

date
echo "抓取 job 信息结束>>>"
```

把以上 shell 脚本命名为 dailyJob.sh，保存在/home/feigu 下，在 crontab 中做如下调用：

```
00 10 * * * sh /home/feigu/dailyJob.sh  >>  /home/feigu/dailyJob.log
```

开始部分的“00 10”表示每天零点 10 分开始运行脚本，Python 爬虫抓取代码中的职位发布

日期要赋值为前一天，也就是每天凌晨都抓取昨日发布的职位。

Hive ETL 部分的代码是用 Java JDBC 实现的，最终代码是做成了 Eclipse 的 Java 工程，打成 JAR 包后，通过命令行“java -jar jarFileName.jar”执行的。因此，同样需要把这个命令行也写在 shell 脚本中，让 crontab 调用。shell 脚本参考代码如下：

```
#!/bin/sh

#######
#  输出 Hadoop 类路径
#######
function exportHadoopClasspath() {
  export CLASSPATH=${HADOOP_CLASSPATH}:${CLASSPATH}
}

#######
#  输出 lib 包下的 jar 类路径
#######
function exportJarClasspath() {
 for loop in `ls /home/feigu/feigu3App/lib/*.jar`;do
   export CLASSPATH=${loop}:${CLASSPATH}
 done
}

#######
#  输出 Hive、HBase 类路径
#######
function exportHiveHbaseClasspath() {
      for loop in `ls /home/hadoop/bigdata/hbase/lib/*.jar`;do
         export CLASSPATH=${loop}:${CLASSPATH}
      done

      for loop in `ls /home/hadoop/bigdata/hive/lib/*.jar`;do
         export CLASSPATH=${loop}:${CLASSPATH}
      done
}

#######
#  把某日目录下所有生成的职位信息导入 Hive
#######
function onedayFile2hive() {
  jobdate="20150501"
  echo "导入$jobdate 日期文件开始>>> "
  #判断指定目录下文件是否已经生成
  jobdata_dir="/home/feigu/jobdata/daily/$jobdate/"
  #判断文件夹是否存在，如果存在，就导入所有文件到 Hive
  if [ -d $jobdata_dir ]; then
```

```
        exportHadoopClasspath
        exportJarClasspath
        exportHiveHbaseClasspath
        echo "导入$jobdata_dir 目录下的文件到 Hive >>> "
        java -cp ${CLASSPATH}:/home/feigu/feigu3App/load2Hive.jar com.feigu.RunDailyHiveJob
$jobdate
    else
      echo "【文件夹不存在】$jobdata_dir"
    fi
}

onedayFile2hive
```

以上 shell 脚本说明如下：

- 开头的三个“export…”函数是把运行 load2Hive.jar 所需要的 JAR 包依赖都引入到类路径中。其中包含了 Hadoop/Hive/HBase 三个组件的 lib 目录下的所有 JAR 包，因为 Hive/HBase 的运行要依赖 Hadoop 的很多 JAR 包，利用 export 命令把所有的 JAR 包都加入到 ${CLASSPATH}中，避免出现 load2Hive.jar 在运行时抛出 ClassNotFoundException 异常。
- “com.feigu.RunDailyHiveJob$jobdate”是运行 Hive ELT 的入口类，调用类中的 main 函数，$jobdate 参数是要进行数据清洗转换的日期分区。
- Python 爬虫抓取的职位结果放在“/home/feigu/jobdata/daily/$jobdate/”目录下，以供 load2Hive.jar 中的“LOAD DATA　LOCAL INPATH”使用，shell 脚本首先判断该目录是否存在。

把上面的 shell 脚本文件保存为/home/feigu/load2Hive.sh，同样配置在 crontab 中即可。执行的开始时间，一定要晚于爬虫执行时间 00:10。两个任务之间存在数据上的依赖关系。

5.10　本章小结

本章重点介绍了 Hadoop 生态系统中数据仓库工具 Hive 的主要特性和使用方法，并且介绍了数据仓库中的一些基本概念。很多初次接触 Hive 的用户都会疑惑：为什么 HiveQL 里面没有更新或删除语法？读者要区分清楚关系数据库和数据仓库之间的差异，才能更好地理解 Hive 的功能定位，以及和其他数据仓库 ETL 工具之间的不同。

从最初的数据建模，到数据加载，再到清洗转换，最后到由各种应用程序使用 Hive 的 API 实现数据调用，本章以生产项目的实际开发步骤为顺序，逐步讲解了 Hive 在不同环节所体现的功能点。由于本书不是专门论述 Hive 使用手册的书籍，因此欲了解更多的功能，还需要读者参考官方文档。

第 6 章
大数据的存储

在上一章中，我们详细介绍了如何利用 Hive 作为 ETL 工具对数据进行加载、清洗、提取维度信息后最终放入聚合表 rpt_job 中。数据维度提取环节，我们定义了 4 个用于检索的维度指标，但是检索的数据来源并不是 rpt_job 表，而是从 HBase 中得到。

Hive 在 Hadoop 生态环境中起到数据仓库的作用，它通过简单的 HQL 调用，实现了后台利用 MapReduce 计算框架对大规模数据的处理，易用性和可靠性是其主要特点。但时效性不是 Hive 的强项，比如一个简单的带 WHERE 条件的 SELECT 语句，相比其他的 RDBMS，执行速度比较慢。另外，Hive 表中的数据也不支持单行数据删除和更新。

在大数据环境下实现低延迟数据读写，就需要用到 HBase，本章就主要讨论 HBase 的一些特性。

6.1 NoSQL 及 HBase 简介

NoSQL，泛指非关系数据库（Not only SQL）。和关系数据库管理系统（RDBMS）相比，NoSQL 不使用 SQL 作为查询语言。其存储可以不需要固定的表模式，通常也会避免使用 RDBMS 的 JOIN 操作，一般都具备水平可扩展的特性。NoSQL 的实现具有两个特征：使用硬盘和把随机存储器作为存储载体。

按照存储格式来分，NoSQL 可以分为 4 类：键值（Key-Value）存储数据库、列存储数据库、文档型数据库和图形（Graph）数据库。目前比较流行的 NoSQL 数据库有 Casssandra、Lucene、Neo4J、MongoDB 和 HBase。

RDBMS 和 NoSQL 的优缺点比较如表 6.1 所示。

表 6.1　RDBMS 和 NoSQL 的优缺点比较

RDBMS 缺点	NoSQL 优点
高并发读写的瓶颈。Web2.0 模式下要实时生成动态页面而无法使用静态化技术，对于每秒上万次的写入 DB 操作，硬盘 I/O 存在明显瓶颈	扩展性强。每种 NoSQL 产品都去掉关系型数据库的关系特性，弱关联的数据更容易扩展，使得很容易实现支撑数据从 TB 级别到 PB 级别的过度。
可扩展性的限制。DB 无法像 Web Server 或 App Server 那样依靠简单增加节点来平滑扩展性能，往往要停机维护和数据迁移。	并发性能好。NoSQL 数据库具有良好的读写性能，其得益于它的弱关系性特点，数据库的结构简单。
事务一致性的负面影响。保证数据完整性的唯一方法是使用事务，这会消耗数据库资源，而很多 Web 系统并不需要严格的数据一致性。	数据模型灵活。NoSQL 无需事先为要存储的数据建立字段，随时可以存储自定义的数据格式。NoSQL 允许用户随时添加字段。而对传统 RDBMS，增删字段是非常麻烦的事情，尤其是对数据量非常大的表。

HBase（Hadoop Database）是一个高可靠、高性能、面向列、可伸缩的分布式数据库系统，它使用类似于 GFS 的 HDFS 作为底层文件存储系统，在其上运行 MapReduce 批量处理数据，使用 ZooKeeper 作为协同服务组件。

HBase 项目使用 Java 语言实现，最初是由 Google Bigtable 原型演化而来的，2007 年第一个简单可用的 HBase 版本发布，2008 年 1 月，Hadoop 升级为 Apache 的顶级项目时，HBase 作为 Hadoop 的子项目而存在。后来随着 Hadoop 版本的提升而不断更新，2010 年 5 月 HBase 成为 Apache 的顶级项目。截至 2014 年年底，HBase 的稳定版本是 0.96。HBase 的运行严重依赖于 Hadoop，且二者的版本存在协调关系。

在 HBase 的概念中，HMaster 主服务器主要负责利用 ZooKeeper 为 Region 服务器分配或移除 Region，起负载均衡的作用。RegionServer 对应于集群中的一个节点，而一个 RegionServer 负责管理多个 Region。一个 Region 代表一个表的一部分数据，所以 HBase 中的一个表可能会需要很多个 Region 来存储其数据，但是每个 Region 中的数据并不是杂乱无章的，HBase 在管理 Region 时会给每个 Region 定义一个 rowkey 的范围，落在特定范围内的数据将交给特定的 Region，从而将负载分摊到多个节点上，充分利用分布式的优点。

HBase 数据存储示意图如图 6.1 所示。

另外，为了防止集群中发生 master 单点故障，HBase 中启动多个 HMaster 实例，通过 ZooKeeper 的投票机制来保证总有一个 HMaster 在运行。

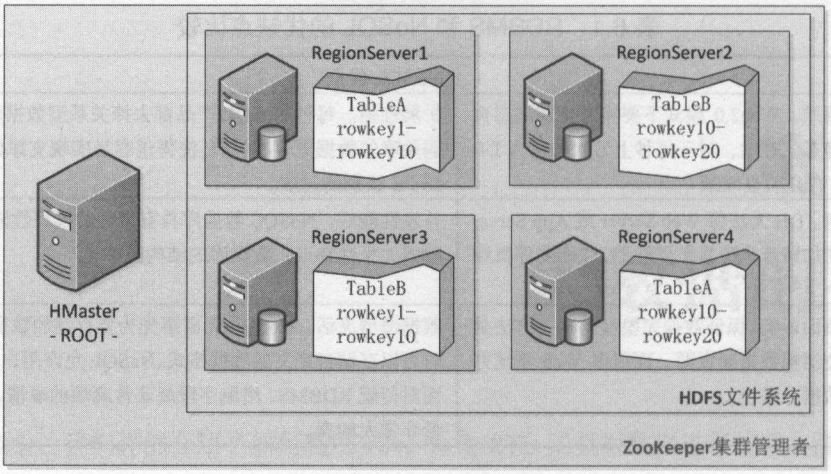


图 6.1　HBase 数据存储示意图

6.2　HBase 中的主要概念

HBase 是一种列式存储的分布式数据库，其核心概念是表（Table）。和传统的数据库一样，表由行和列组成，但 HBase 把不同的列组成一个列族（Column Family），同一列的值又可以有多个版本（version）。这种组织形式主要是出于存取性能的考虑。

表：在 HBase 中数据以表的形式存储。表名使用 Java String 类型或 byte[]字节数组来表示。每个表名对应 HDFS 中的一个目录结构。创建表时需要指定表名和至少一个列族。列族影响表的物理存储结构。

列族：一些列的集合。列族创建好后不能频繁修改，数量不能太多。列族名由可见的字符组成。列族中包含列的数量没有限制，可以有数百万个列。列值没有类型和长度的限定。一个列族中的所有列存储在同一个底层文件(HFile)中。常见的引用列的格式是 family:qualifier，其中 qualifier 是任意的字节数组。

行键：HBase 中最重要的概念之一，它在表中用来唯一地标示一行，其值保存为二进制字节数组。表中的行是按照行键的字典序排列存储的。行键存储示例如图 6.2 所示，其中第一列以“row-”开头的是表中每一行的行键。

在 HBase 中行键是默认索引，所以在设计行键时必须考虑检索数据的效率。比如把常用的检索字段组合作为 rowkey，并且要保证其唯一性，但 rowkey 长度又不能太长，太长的 rowkey 会增加存储开销，降低内存利用率，从而降低索引命中率。另外，rowkey 的设计同时要考虑如何使其

值能够尽量散列，这样会保证所有的数据不都在一个 Region 上，避免进行读写时负载集中在个别 Region 上。

```
hbase(main):001:0> scan 'table1'
ROW                   COLUMN+CELL
 row-1                column=cf:val, timestamp=1420013446529, value=value-1
 row-10               column=cf:val, timestamp=1420013561396, value=value-10
 row-11               column=cf:val, timestamp=1420013570461, value=value-11
 row-1a               column=cf:val, timestamp=1420013605692, value=value-1a
 row-2                column=cf:val, timestamp=1420013577217, value=value-2
 row-22               column=cf:val, timestamp=1420013584822, value=value-22
 row-3                column=cf:val, timestamp=1420013592073, value=value-3
 row-abc              column=cf:val, timestamp=1420013599133, value=value-abc
8 row(s) in 1.2410 seconds
```

图 6.2　行键存储示例

单元：行键、列族和列名一起确定了一个单元。存储在单元里的数据称为单元值（Value）。单元值没有数据类型，存储为字节数组 byte[]。

时间版本：单元值有时间版本，时间版本用时间戳标识，是一个 long 型数字，默认使用当前时间戳。HBase 保留单元值时间版本的数量基于列族进行配置，默认数量是 3 个。图 6.3 展示了如何保存两个不同版本的数据。

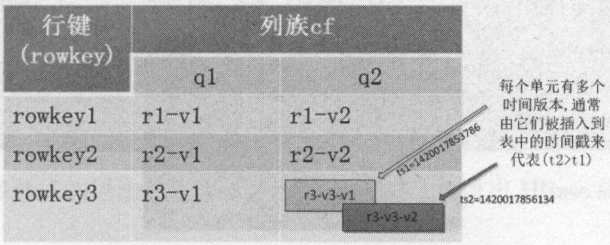


图 6.3　保存两个不同版本的数据示例

6.3　HBase 客户端及 JavaAPI

使用 HBase 客户端可以方便地操作 HBase 表中的数据。其中，HBase shell 工具、原生 Java API 客户端和 Thrift 客户端是三种最常见的工具。此外，还有 REST（Representational State Transfer，表述性状态转移）和 Avro。

HBase shell 是安装 HBase 时就已经自带的客户端工具，它可以接收用户输入的 DDL 和 DML 语句来操作 HBase 表中的数据，其具体实现是用 Ruby 语言编写的，并使用 JRuby 解释器。它有两种使用方式：交互模式和批量模式。

和 Hive CLI 界面相似，HBase 也提供了控制台客户端，可以操作 HBase 中的数据。客户端工

具在${HBASE_HOME}\bin 目录下，通过输入 hbase shell 命令进入：

```
hadoop@slave02:~/bigdata/hbase/bin$ hbase shell
2015-08-17 14:43:33,811 INFO  [main] Configuration.deprecation: hadoop.native.lib is deprecated.
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 0.98.5-hadoop2, rUnknown, Mon Aug  4 23:58:06 PDT 2014

hbase(main):001:0>
```

查看 HBase 集群状态使用 status 命令，可以看到有两个 HMaster：

```
hbase(main):001:0> status
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/hadoop/bigdata/hbase/lib/slf4j-log4j12-1.6.4.
SLF4J: Found binding in [jar:file:/home/hadoop/bigdata/hadoop/share/hadoop/common/lib
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
2 servers, 0 dead, 2.0000 average load
```

查看 HBase 版本使用 version 命令：

```
hbase(main):002:0> version
0.98.5-hadoop2, rUnknown, Mon Aug  4 23:58:06 PDT 2014
```

查看 HBase 的所有表使用 list 命令，目前有两个表：

```
hbase(main):003:0> list
TABLE
rpt_job
test1
2 row(s) in 0.1890 seconds

=> ["rpt_job", "test1"]
```

在 shell 工具中操作表的命令包括 DDL 和 DML。DDL 是数据定义语言，包括创建表、修改表、上线和下线表、删除表等。创建表使用 create 命令：

```
hbase(main):008:0> create 'test1', {NAME => 'cf', VERSIONS => 3 , COMPRESSION => 'GZ' }
0 row(s) in 1.0450 seconds

=> Hbase::Table - test1
```

上面命令在 HBase 中创建了表“test1”，同时创建了一个列族“cf”，“VERSION=>3”的含义是每个单元格都保存 3 个版本的值，表的 HFile 文件采用“GZ”压缩方式。在创建表时，并不需要指定列族下的列名。列名是在插入数据时才创建的。

DML 是数据操作语言，包括数据写入、删除、查询和清空，不包括更新操作。

put 命令按行写入数据，如下所示：

```
hbase(main):010:0* put 'test1','rowkey1', 'cf:q1', 'r1-v1'
0 row(s) in 0.1970 seconds

hbase(main):011:0> put 'test1','rowkey1', 'cf:q2', 'r1-v2'
0 row(s) in 0.0080 seconds

hbase(main):012:0> put 'test1','rowkey2', 'cf:q1', 'r2-v1'
0 row(s) in 0.0060 seconds

hbase(main):013:0> put 'test1','rowkey2', 'cf:q2', 'r2-v2'
0 row(s) in 0.0060 seconds

hbase(main):014:0> put 'test1','rowkey2', 'cf:q3', 'r2-v3'
0 row(s) in 0.0160 seconds
```

上面命令在 test1 表中插入了两条数据，行键分别是“rowkey1”和“rowkey2”，第一行包含两列，第二行包含三列。

scan 命令用于检索表数据，可以指定列族中的特定列、显示某个列的所有时间戳版本或限定返回的记录行数。最简单的语法格式是“scan 表名”，如下所示：

```
hbase(main):016:0* scan 'test1'
ROW                                COLUMN+CELL
 rowkey1                           column=cf:q1, timestamp=1439795672137, value=r1-v1
 rowkey1                           column=cf:q2, timestamp=1439795672209, value=r1-v2
 rowkey2                           column=cf:q1, timestamp=1439795672243, value=r2-v1
 rowkey2                           column=cf:q2, timestamp=1439795672276, value=r2-v2
 rowkey2                           column=cf:q3, timestamp=1439795673735, value=r2-v3
2 row(s) in 0.0500 seconds
```

关于 scan 命令更多形式的语法组合，可以参考相关手册。退出 HBase shell 客户端，使用 exit 命令。

HBase 官方代码中包含了原生访问客户端，是由 Java 语言实现的，这也是最直接、最高效的客户端。相关类都在 org.apache.hadoop.hbase.client 包中，涵盖增、删、改、查等 API。主要类包括 HTable、HBaseAdmin、Put、Get、Scan、Delete 等。

在使用原生客户端之前，首先要配置客户端，在 HBaseConfiguration 类中创建一个配置实例，让该实例依次加载所依赖的 Hadoop 和 HBase 中的多个配置文件。示例代码如下：

```
Configuration conf = HBaseConfiguration.create();
conf.set("dfs.permissions", "false");
// 从本地加载配置文件
String hadoop_home = System.getenv("HADOOP_HOME");
String hhase_home = System.getenv("HBASE_HOME");
conf.addResource(new Path(hadoop_home + "/conf/mapred-site.xml"));
conf.addResource(new Path(hadoop_home + "/conf/core-site.xml"));
conf.addResource(new Path(hhase_home + "/conf/hbase-site.xml"));
conf.addResource(new Path(hadoop_home + "/conf/yarn-site.xml"));
```

HTable 类中的方法分为 4 类：增删查方法、获取表元数据方法、获取状态信息方法和设置属性方法。HTable 中的 put 方法可以向表中逐条或批量插入数据。下面代码演示了如何在刚才创建的 test1 表中插入 3 行数据。

```
public static void insertTableTest(Configuration conf ) {
   try {
     String tableName = "test1";
     HTable table = new HTable(conf, tableName);
     for(int i=1; i<=3; i++) {
       Put put = new Put(Bytes.toBytes("rowkey"+i));
put.add(Bytes.toBytes("cf"),Bytes.toBytes("q1"), Bytes.toBytes("r"+i+"v1"));
put.add(Bytes.toBytes("cf"),Bytes.toBytes("q2"), Bytes.toBytes("r"+i+"v2"));
       table.put(put);
     }
     table.close();
}
catch… …
}
```

使用 HBaseAdmin 类中的 getTableDescriptor 方法可以获取表的元数据信息。使用 HTable 类中的 get 方法和 scan 方法可以检索表中的数据。get 方法实现指定 rowkey 检索，scan 方法实现组合条件检索，scan 和 FilterList 结合可以实现各种组合条件检索。有关 HBase Java API 的更多功能，可以参考《HBase 权威指南》（人民邮电出版社出版）。

6.4 Hive 数据导入 HBase 的两种方案

Hive 作为一个数据仓库的客户端工具，本身是不保存数据的，它所操作的表数据都存放在 HDFS 中。而在 HBase 中创建表、插入的数据是存放在 HBase 数据库内部的，这种底层文件叫作 HFile，它被看作是 HDFS 的子集，使用 hadoop 命令可以看到这个文件。执行如下命令，可以看到目前 HBase 中有两个表，表名即目录名：

```
hadoop@slave02:~/bigdata$ hadoop fs -ls /user/hadoop/hbase/data/default
Found 2 items
drwxr-xr-x   - hadoop supergroup          0 2015-05-27 17:49 /user/hadoop/hbase/data/default/rpt_job
drwxr-xr-x   - hadoop supergroup          0 2015-08-17 15:07 /user/hadoop/hbase/data/default/test1
```

Hive 和 HBase 整合有两种方案，分别是利用 hive_hbase-handler.jar 工具类和编写 MapReduce 算法把 Hive 中的数据导入 HBase 中。

6.4.1 利用既有的 JAR 包实现整合

在${HIVE_HOME}\lib 目录下有 hive_hbase-handler.jar 文件，利用该 JAR 包提供的功能可以实现 Hive 和 HBase 之间的数据互通。数据是保存在 HBase 中的，而 Hive 作为一个连接客户端可

以读写 HBase 表中的数据。首先要在 hive-site.xml 中配置 hive.aux.jars.path 属性，值指定为 hive-hbase-handler.jar。代码如下：

```
<property>
   <name>hive.aux.jars.path</name>
   <value>file:///home/hadoop/bigdata/hive/lib/hive-hbase-handler-0.13.0.jar,
file:///home/hadoop/bigdata/hive/lib/zookeeper-3.4.5.jar,file:///home/hadoop/bigdat
a/hbase/lib/protobuf-java-2.5.0.jar,file:///home/hadoop/bigdata/hbase/lib/hbase-com
mon-0.96.2-hadoop2.jar</value>
      <description>add by jars for java connect hive or hbase</description>
</property>
```

在 Hive 中创建城市表 hbase_table_city，它包含两个字段，分别是城市等级和名称。建表语句如下：

```
CREATE TABLE  hbase_table_city(level int, city string)
 STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
 WITH SERDEPROPERTIES ("hbase.columns.mapping" = ":key,cf1:val")
 TBLPROPERTIES ("hbase.table.name" = "t_city");
```

HBase 中的表名叫 t_city, Hive 表中的 level 作为 HBase 中的 rowkey, city 的值对应 HBase 表中的列族“cf1:val”。

在 Hive 中运行脚本，如下所示：

```
hive> CREATE TABLE hbase_table_city(level int, city string)
    >  STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
    >  WITH SERDEPROPERTIES ("hbase.columns.mapping" = ":key,cf1:val")
    >  TBLPROPERTIES ("hbase.table.name" = "t_city");
OK
Time taken: 8.805 seconds
hive> select * from hbase_table_city;
OK
Time taken: 1.427 seconds
```

然后查看 HBase 数据库，发现也创建了表 t_city，如下所示：

```
hbase(main):001:0> list
TABLE
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/hadoop/bigdata/hbase/lib/sl
SLF4J: Found binding in [jar:file:/home/hadoop/bigdata/hadoop/share
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an
2015-08-17 16:19:34,332 WARN  [main] util.NativeCodeLoader: Unable
able
rpt_job
t_city
2 row(s) in 7.5000 seconds

=> ["rpt_job", "t_city"]
hbase(main):002:0> scan 't_city'
ROW                                    COLUMN+CELL
0 row(s) in 0.1240 seconds
```

在 HBase 的 t_city 表中插入 3 条测试数：

```
hbase(main):003:0> put 't_city', 1, 'cf1:val', 'Beijing'
0 row(s) in 0.4960 seconds

hbase(main):004:0> put 't_city', 2, 'cf1:val', 'Shanghai'
0 row(s) in 0.0640 seconds

hbase(main):005:0> put 't_city', 3, 'cf1:val', 'Guangzhou'
0 row(s) in 0.0420 seconds

hbase(main):006:0> scan 't_city'
ROW                                    COLUMN+CELL
 1                                     column=cf1:val, timestamp=1439799990571, value=Beijing
 2                                     column=cf1:val, timestamp=1439799999008, value=Shanghai
 3                                     column=cf1:val, timestamp=1439800008336, value=Guangzhou
3 row(s) in 0.1730 seconds
```

在 Hive 表中也会看到相应的 3 条数据：

```
hive> SELECT * FROM hbase_table_city;
OK
1       Beijing
2       Shanghai
3       Guangzhou
Time taken: 0.372 seconds, Fetched: 3 row(s)
```

6.4.2 手动编写 MapReduce 程序

使用 hive_hbase-handler.jar 工具类虽然很方便，但不是很灵活。如果想实现把 Hive 的 rpt_job 表中每个日期分区下的数据导入 HBase，还是采用代码方式实现更加灵活。

首先在 HBase 中创建表 rpt_job，DDL 语句只有一行，创建一个列族“cf”。

```
hbase(main):006:0> create 'rpt_job', 'cf'
```

将数据从 Hive 导入到 HBase 中还有一种方法，就是利用 HBase 所提供的工具类 TableMapReduceUtil 来实现。该类在 hadoop.hbase.mapreduce 包下，使用其中包含的 TableMapper 和 TableReducer 类来读写表中的数据。

TableMapReduceUtil 类把从 Hive 表中读取数据作为 Mapper 计算模型，写入数据到 HBase 作为 Reducer 模型。首先指定要读取的 Hive 表中的列名和 HBase 表中的 rowkey，作为 Mapper 的输出。

在 TableReducer 类中重写方法 reduce()，Text 类型的 key 作为 HBase 表的行键 rowkey，values 枚举列表中存放的是每个列族中的值，把这两者构造成一个 Put 对象实例，放入 reduce 方法的输出 value 中，而 reduce 方法的输出 key 是 HBase 中的表名。

将数据从 Hive 导入到 HBase 中，关键点在于如何从 Hive 表中选出合适的列，组合成 HBase 表的行键，并把 Hive 表中的其他字段作为列族插入到 HBase 表中。HBase 表中的行键，和 RDBMS 中的主键有类似之处。行键的作用主要有二：其一，rowkey 是 HBase 的 KeyValue 存储结构中的 key，作为一条记录的唯一标识；其二,由于行键是以字典顺序从小到大排列存储的，并且 HBase 要在 rowkey 上建立索引，按照 rowkey 对数据进行检索效率是非常高的，因此在设计行键时，要把 Hive 表中经常用到的检索列组合起来作为行键。

在如图 6.4 所示的 Web 检索页面中，我们需要把 Hive 表中的职位发布日期、职位名称、公司名称、月薪范围、工作年限、学历要求和工作地点组合成 HBase 表中的 rowkey，而把其他字段作为列族来存储，如图 6.5 所示。

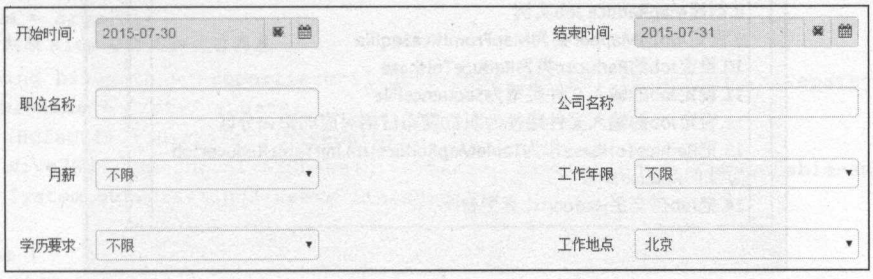


图 6.4　职位 Web 检索页面

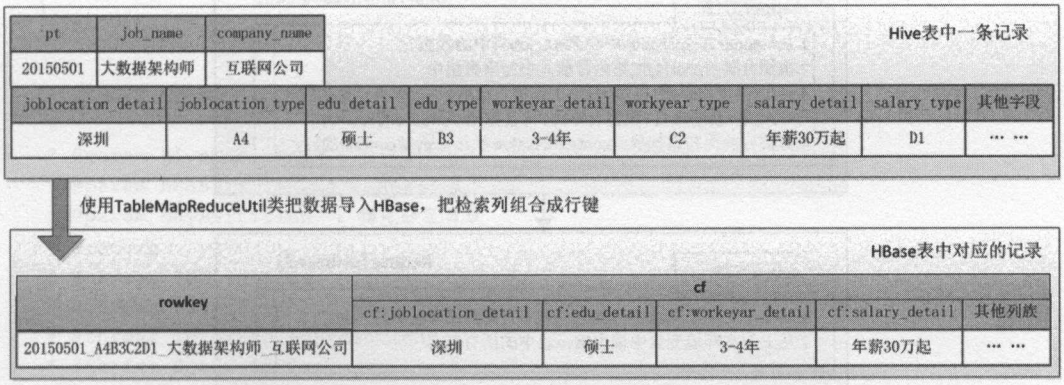


图 6.5　Hive 到 HBase 数据存储示意图

rowkey 的存储内容是“20150501_A4B3C2D1_大数据架构师_互联网公司”，格式为“职位发布日期_四个维度值_职位名称_公司名称”。之所以这样设计，是考虑到检索 rowkey 时方便代码处理，具体情况在 6.5.2 节中会做详细说明。

使用 TableMapReduceUtil 类把数据导入 HBase 可以分为三步，分别由 Hive2HbaseRptJob、MapFromHiveSeqfile 和 ReduceToHbase 三个类来完成。每个类中的处理逻辑如图 6.6 所示。

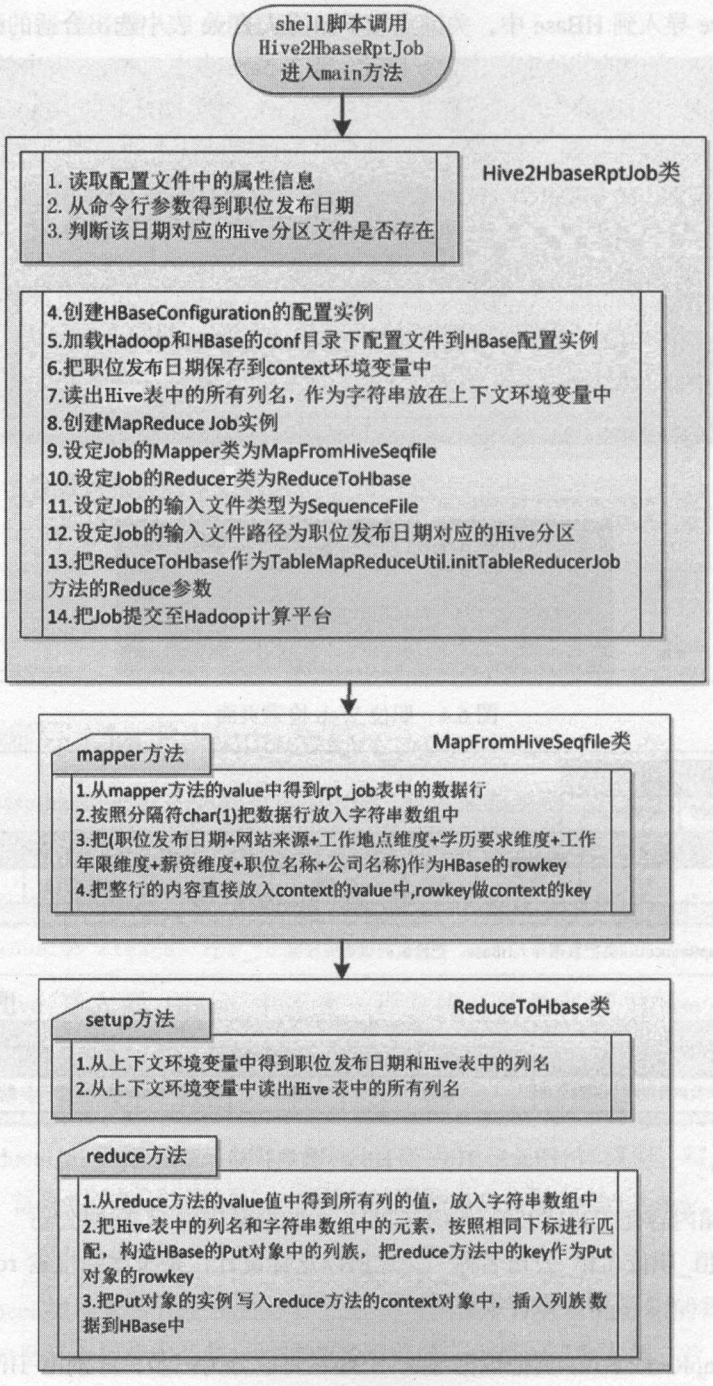


图 6.6　三个类的处理逻辑图

Hive2HbaseRptJob 类代码如下：

```
public  static void main(String[] args) throws IOException, InterruptedException,
ClassNotFoundException,
    SQLException {
   PropertiesUtil.getPropertyFile("feigu3.properties"); // 读取本地同路径下的配置文件
   String hivetablename = PropertiesUtil.getPropertyValue("hivetablename") ;
   String hbasetablename = PropertiesUtil.getPropertyValue("hbasetablename") ;
   String hiveDbname = PropertiesUtil.getPropertyValue("hivedbname") ;
   String date = ""; // 从控制台读入日期，格式为 yyyymmdd
   if (args.length==0){
    System.out.println("input date empty!");
    return;
   }
   date = args[0];
   // 判断 Hive 分区文件是否存在
   String hivepath = PropertiesUtil.getPropertyValue("fs.default.hivepath") + "/" +
hivetablename + "/pt=" + date;
   if (HdfsUtil.isDirExists(hivepath)){
      Hive2HbaseRptJob.hive2Hbase(hiveDbname, hivetablename, hbasetablename, date);
      System.out.println("Insert finished!");
   }
   else {
    System.out.println("【hive path doesn't exist!】"+hivepath);
   }
 }

 /**
  * 将 HDFS 文件数据导入到 HBase 的接口
  *
  * @param hiveTableName : Hive 表名
  * @param hbaseTableName : HBase 表名
  * @param dayPartition  : 指定分区目录
  * @return
  * @throws IOException
  * @throws ClassNotFoundException
  * @throws InterruptedException
  * @throws SQLException
  */
 public static boolean hive2Hbase(String hiveDbName, String hiveTableName, String
hbaseTableName, String dayPartition)
    throws IOException, InterruptedException, ClassNotFoundException, SQLException {

   Configuration conf = HBaseConfiguration.create();
   conf.set("dfs.permissions", "false");
   // 从本地加载
   String hadoop_home = System.getenv("HADOOP_HOME");
   String hhase_home = System.getenv("HBASE_HOME");
```

```
    conf.addResource(new Path(hadoop_home + "/conf/mapred-site.xml"));
    conf.addResource(new Path(hadoop_home + "/conf/core-site.xml"));
    conf.addResource(new Path(hhase_home + "/conf/hbase-site.xml"));
    conf.addResource(new Path(hadoop_home + "/conf/yarn-site.xml"));

    conf.set("hbasetable", hbaseTableName);
conf.set("job_date", dayPartition);
    // 把 Hive 中的所有列都导入 HBase 表
    String hivetablecolumnlist = ""; // 把所有列表存储在字符串中，以逗号分隔
    List<String> colNameLst = HiveJdbcUtil.getTableColumn(PropertiesUtil.getPropertyValue
("hive.jdbcurl"),     hiveDbName+"."+hiveTableName); // 获取列名
    for (int i = 0; i < colNameLst.size(); i++) {
      if (i == colNameLst.size() - 1) {
        hivetablecolumnlist = hivetablecolumnlist + colNameLst.get(i);
      }
      else {
        hivetablecolumnlist = hivetablecolumnlist + colNameLst.get(i) + ",";
      }
    }

    conf.set("hivetablecolumnlist", hivetablecolumnlist);

    Job job = Job.getInstance(conf, "Hive2Hbase rpt_job ");
    job.setJarByClass(Hive2HbaseRptJob.class);

    job.setMapperClass(MapFromHiveSeqfile.class);
    job.setMapOutputKeyClass(Text.class);
    job.setMapOutputValueClass(Text.class);
    // Hive 中的 rpt_job 表是以 SequenceFile 格式存储的
    job.setInputFormatClass (SequenceFileAsTextInputFormat.class);

    Path in = new Path(PropertiesUtil.getPropertyValue("fs.default.hivepath") + "/" +
hiveTableName + "/pt="+ dayPartition);
    SequenceFileAsTextInputFormat.addInputPath(job, in);

    job.setReducerClass(ReduceToHbase.class);
    TableMapReduceUtil.initTableReducerJob(hbaseTableName, ReduceToHbase.class, job);
    return job.waitForCompletion(true);
  }
```

MapFromHiveSeqfile 类代码如下：

```
  * Hive 的 SequenceFile 作为输入的 Mapper 类
  */
 public static class MapFromHiveSeqfile extends Mapper<Text, Text, Text, Text> {

    // 职位发布日期，pt 的内容
    private String job_date = "";
```

```
    @Override
    Protected  void setup(Context context) throws IOException, InterruptedException {
      Configuration conf = context.getConfiguration();
      job_date = conf.get("job_date");
    }

    @Override
    public void map(Text key, Text values, Context context) throws IOException,
InterruptedException {
      String[] valueArray = values.toString().split(String.valueOf(_asc1), -1);
      // 把（职位发布日期+网站来源+工作地点+学历要求+工作年限+薪资+职位名称+公司名称）作为 rowkey
      String rowkey = "";
      if (valueArray.length == 25) {
        rowkey += job_date + "_";  // 职位发布日期
        rowkey += valueArray[1];  // 网站来源
        rowkey += valueArray[5];  // 工作地点
        rowkey += valueArray[8];  // 学历要求
        rowkey += valueArray[13];  // 工作年限
        rowkey += valueArray[15];  // 薪资
        rowkey += "_" + valueArray[3];  // 职位名称
        rowkey += "_" + valueArray[16];  // 公司名称
      }
      // 把整行内容直接放入 value 中
      context.write(new Text(rowkey), values);
    }
  }
```

ReduceToHbase 类代码如下：

```
  /**
   * 将 map 结果存入 HBase 的 Reducer 类
   */
  public static class ReduceToHbase extends TableReducer<Text, Text, Text>{

    private List<String> hiveColumnLst = new ArrayList<String>();
    private String hbasetablename = "";
    private String family = "cf";
    // 职位发布日期，pt 的内容
    private String job_date = "";

    @Override
    Protected  void setup(Context context) throws IOException, InterruptedException {
      Configuration conf = context.getConfiguration();
      hbasetablename = conf.get("hbasetable");
      job_date = conf.get("job_date");
      String hivetablecolumnlist = conf.get("hivetablecolumnlist");
      for (String x : hivetablecolumnlist.split(",")) {
        hiveColumnLst.add(x);
```

```
        }
      }
      @Override
      public void reduce(Text key, Iterable<Text> values, Context context) throws
  IOException, InterruptedException {
        for (Text val : values) {
          String[] valArray = val.toString().split(String.valueOf(_asc1));
          // valArray 中不包含 pt 分区列，而 hiveColumnLst 中包含 pt 分区列，两者相差 1
          if (valArray.length == hiveColumnLst.size()-1) {
            byte[] b_rowkey = Bytes.toBytes(key.toString());
            byte[] b_family = Bytes.toBytes(family);
            for (int x=0; x<valArray.length; x++) {
              Put put = new Put(b_rowkey); // rowkey
              String each_val = valArray[x];

              byte[] b_qualifier = Bytes.toBytes(hiveColumnLst.get(x));
              byte[] b_val = Bytes.toBytes(each_val);
              put.add(b_family, b_qualifier, b_val);
              context.write(new Text(hbasetablename), put);
            }
          }
        }
      }
    }
```

6.5 使用 Java API 查询 HBase 中的职位信息

上一节中，利用 TableMapReduceUtil 工具类把 Hive 表中的数据导入了 HBase。通过爬虫抓取的职位信息，最终是要在检索页面中可以由用户任意查询的。既然是检索，只要有一个数据来源即可，那么为什么不从 Hive 的 rpt_job 表中检索，而要利用 HBase 呢？在 HBase 中又是如何实现对上面我们抽取的 4 个维度进行检索的？本节就回答了这些问题。

6.5.1 为什么是 HBase 而非 Hive

Hive 在 Hadoop 生态系统中的定位是数据仓库工具，处理数据而不存储数据。由于 HQL 在运行时会转换成 MapReduce 任务，利用 Hadoop 分布式计算框架，得以可靠地完成大规模数据运算。但 Hive 自身也有不擅长的地方，比如无法实现单条数据的更新；HQL 的 SELECT 语句运行是相当慢的。

传统的 RDBMS，比如 Oracle，在服务器接收到客户端提交的 SQL 后，会依次进行语法解析、创建执行计划、执行 SQL、返回结果集到客户端等工作。而在 Hive CLI 界面中执行一条包含 WHERE 子句的 SELECT 语句时，在非“本地模式”下，Hive 会把 HQL 转换成 MapReduce Job，

提交到 Hadoop 的 Master（JobTracker）节点，由该节点向各个 Slave 节点分发任务，每个 Slave 节点又会启动数量不等的 TaskTracker 进程，每个 TaskTracker 进程都会启动一个 JVM。运算结束后，每个节点上的数据又要进行汇总，最后返回到客户端。即使返回的结果集只有一行数据，这种任务分配的模式也是不变的。

Hive 查询适用的场合是在不需要实时响应的后台任务模式下，比如每天晚上从 Hive 表中抽取数据，生成汇总报表的定时任务。而如果是在提供检索服务，需要及时把查询结果呈现给用户的场合下，使用 Hive 就不合适了。

HBase 可以解决检索效率问题。在 6.4.2 节中我们把 Hive 表中的数据原封不动地导入到了 HBase 中，目的就是为了利用 HBase 实现实时数据检索，增强用户体验；而把 Hive 作为资料库完成对实时性要求不高的数据统计、报表展示功能。

6.5.2　多条件组合查询 HBase 中的职位信息

使用 Java API 实现 HBase 数据检索有三种途径，分别是使用 HTable.get()方法、Scan（扫描器）类和 Filter（过滤器）类。

- HTable.get()方法最简单，它根据行键的值获取表中对应的数据，类似于在 RDBMS 中使用主键获取整条记录的功能（6.6 节中会有具体示例）。
- Scan 类提供了简单的对表进行数据扫描、分块的功能，使用方法和 get()类似。get()方法每次得到一条数据，而 Scan 类得到的是数据集合。利用 Scan 类可以得到表中指定范围的某块记录，使用参数 startRow 来定义扫描读取 HBase 表的起始行键，stopRow 参数来限定结束行键。此外，Scan 类还允许只返回某些列族、一个列族中的某些列值，类似于 SQL 中的投影功能。
- Filter 类，顾名思义，是用来描述检索条件的。它提供了针对行键、列族、版本号的多样化筛选功能，其设计思想来源于 QBE（Query By Example）。并且通过多个 Filter 条件的组合还可以模拟 RDBMS 中的复合条件检索功能，作为描述数据筛选的条件，Filter 类可以和 get()方法或 Scan 类配合使用。

在职位检索页面中，用户可以选择的筛选条件有很多，这些条件之间是“AND”关系，在代码实现上使用了 Scan 类配合 Filter 的方法。HBase 中提供的过滤器多达十几种，常用的有行键过滤器（RowFilter）、列名过滤器（QualifierFilter）、值过滤器（ValueFilter）、单列值过滤器（SingleColumnValueFilter）等。由于 HBase 中的数据行是按照行键（rowkey）做索引的，基于 RowFilter 的数据筛选往往是效率最高的。

在 6.4.2 节中，招聘职位数据在进入 HBase 表时，rowkey 的存储格式为“职位发布日期_4 个

维度值_职位名称_公司名称”，比如“20150501_A4B3C2D1_大数据架构师_互联网公司”。这里把 Web 页面中每个检索条件的值都作为行键的一部分，来提高检索效率。

有读者可能会问，既然使用行键检索速度快，那么直接把 rpt_job 表中的所有字段都拼在一起作为行键，岂不是最方便了？比如想实现对“职位描述”和“公司简介”这两个内容较多的字段做模糊检索，放入行键中不是也可以吗？HBase 表的 rowkey 设计要遵循一定的原则：首先，rowkey 的值要保证唯一，如果行键的值重复，列族中的值会覆盖已有的内容；其次，rowkeyr 的值的长度不能太长，否则会影响性能；最后，rowkey 的值应该做到尽量散列，以防止数据都集中在某个 DataNode 上，无法充分利用集群的优势。关于 rowkey 设计的原则和技巧，可以参考《HBase 权威指南》（人民邮电出版社出版）中“9.1 行键设计”章节。

继续 Filter 检索的话题，由于对行键格式“职位发布日期_4 个维度值_职位名称_公司名称”中不同部分使用不同的 Java API 语法来实现检索，因此下面把代码拆开进行讲解。

1. 对时间段范围进行筛选

```
HTable table=new HTable(conf, tablename);
FilterList filterList = new FilterList(FilterList.Operator.MUST_PASS_ALL);
Scan scan = new Scan();

// 职位发布日期介于开始时间和结束时间之间（检索列是 rowkey）
if (!("".equals(sd)&& "".equals(ed))) {
   //  因为 rowkey 的格式是 20150320_A1B1C2D9，所以要加上分隔符_
   sd = sd + "_";
   ed = ed + "_";
   filterList.addFilter(
      new RowFilter(
         CompareFilter.CompareOp.GREATER_OR_EQUAL,
         new BinaryPrefixComparator(Bytes.toBytes(sd))));

   filterList.addFilter(
      new RowFilter(
         CompareFilter.CompareOp.LESS_OR_EQUAL,
         new BinaryPrefixComparator(Bytes.toBytes(ed))));
}
```

上面代码中出现了 HBase API 中的几个类，分别说明如下。

（1）HTable

HTable 是 HBase 提供的主要客户端接口，在 org.apache.hadoop.hbase.client 包下。通过这个类，用户可以完成向 HBase 存储和检索数据，或者删除数据等操作。创建一次 HTable 实例会消耗一定的系统资源，因此代码中最好使用单例模式。HTable 不是线程安全的，如果需要为每个线程都创建一个实例，则可以使用 HTablePool 类。

（2）RowFilter

RowFilter（行过滤器）用来描述行键的过滤条件。创建 RowFilter 实例时，在构造方法中要指定两个参数，分别是比较运算符和比较器。创建实例后，可以把它赋值给 Scan 类，比如 Scan.setFilter(myRowFilter)，也可以把它追加到过滤器链，比如 FilterList.addFilter(myRowFilter)。

（3）FilterList

在上面代码中，我们把两个 RowFilter 实例添加到了 FilterList 中。FilterList 适用于需要多个过滤器共同限制返回到客户端的数据的情况。类似于写 SQL 时的 WHERE 子句，每个 AND（OR）条件是一个单独的 Filter，拼接在一起就是一个 FilterList。创建 FilterList 时可以在构造方法中指明 Operator 参数，该参数决定了各个 Filter 间的组合结果。FilterList.Operator 有两个可选枚举值，如表 6.2 所示。

表 6.2　Filter.Operator 的两个可选枚举值

值类型	描述
MUST_PASS_ALL	当所有过滤器都允许包含这个值时，该值才会出现在结果集中。这是 FilterList 构造方法的默认值
MUST_PASS_ONE	只要有一个过滤器包含这个值，该值就会出现在结果集中

（4）CompareFilter

使用比较过滤器（CompareFilter）可以实现对行键值、列族名、列名、列值等 HBase 表结构中的各个部分进行检索，它包含的子类有 RowFilter、FamilyFilter、QualifierFilter 和 ValueFilter 等。使用比较过滤器对行键值和给定值进行比较时，要区分出两个值的比较原则，即“大于”、“小于”、“等于”、“大于或等于”和“小于或等于”等。这些比较原则在 HBase 中有一个专有的名称——“比较运算符”（CompareFilter.CompareOp），比较运算符的取值使用枚举值类型，一共包括 7 种类型，如表 6.3 所示。

表 6.3　比较运算符的 7 种取值

值类型	描述
LESS	匹配小于给定值的行键
LESS_OR_EQUAL	匹配小于或等于给定值的行键
EQUAL	匹配等于给定值的行键
NOT_EQUAL	匹配不等于给定值的行键
GREATER_OR_EQUAL	匹配大于或等于给定值的行键
GREATER	匹配大于给定值的行键
NO_OP	排除一切行键值

在上面的代码中，我们使用“大于或等于开始时间并且小于或等于结束时间”限定了招聘职

位发布的起止时间段。

（5）BinaryPrefixComparator

比较过滤器（包括行键过滤器）在创建时，除了需要比较运算符之外，还需要一个参数——比较器（Comparator）。比较器提供了多种方法来比较不同的键值。HBase 对基于 CompareFilter 的过滤器提供的比较器如表 6.4 所示。

表 6.4　基于 CompareFilter 的比较器

比较器类型	描述
BinaryComparator	把行键值和给定值转换成字节数组后，调用 Bytes.compareTo()方法进行比较
BinaryPrefixComparator	与上面的类似，只是从左端开始前缀匹配
NullComparator	只判断当前行键值是否为 null
BitComparator	通过 BitwiseOp 类提供的位操作符，执行字节位的比较
RegexStringComparator	根据一个正则表达式来比较行键中的值
SubstringComparator	比较给定的值是否在行键中出现过，即给定的值是否是行键的子串

在上面的代码中，使用 BinaryPrefixComparator 构造了开始和结束两个行键过滤器。因为在 HBase 的 rpt_job 表中，行键的值都是以“YYYYMMDD_”开头的，所以使用 BinaryPrefixComparator 正好满足要求。并且，由于 HBase 表中的数据是按照行键值的字典序排列的，经过“大于或等于开始时间并且小于或等于结束时间”和“从左端前缀匹配”的筛选，就可以定位到所需要的数据集。按照职位发布时间范围筛选数据的处理过程如图 6.7 所示。

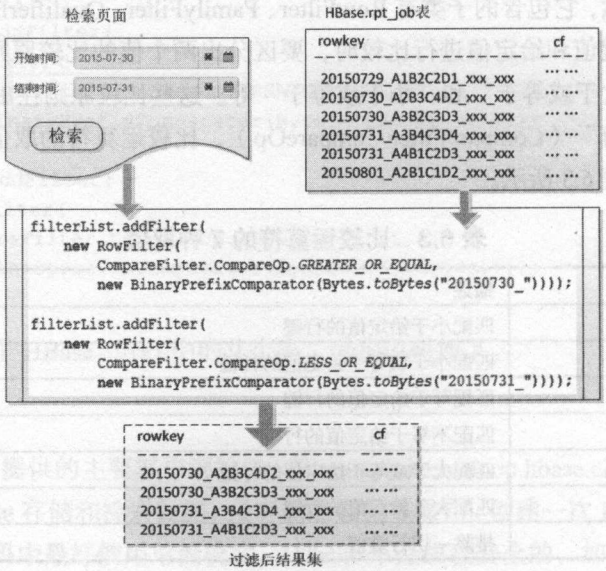


图 6.7　按照职位发布时间筛选数据的处理过程

2. 对 4 个维度和职位名称进行筛选

上面的代码只是针对时间段进行筛选，下面的代码片段是对 4 个维度和职位名称进行字符串匹配检索。

```
// 薪资维度
if (!"00".equals(salary)) {
filterList.addFilter(
    new RowFilter(
    CompareFilter.CompareOp.EQUAL,
    new SubstringComparator(salary)));
}
// 工作年限维度
if (!"00".equals(work_year)) {
filterList.addFilter(
    new RowFilter(
    CompareFilter.CompareOp.EQUAL,
    new SubstringComparator(work_year)));
}
// 职位名称模糊检索
if (!"".equals(job_name)) {
filterList.addFilter(
    new RowFilter(
    CompareFilter.CompareOp.EQUAL,
    new SubstringComparator(job_name)));
}
scan.setFilter(filterList);
......
```

由于对 4 个维度值和对职位名称、公司名称检索的处理方式相同，所以上面的代码片段只列出了针对薪资维度、工作年限维度和职位名称模糊检索的处理逻辑。在检索页面中，将 4 个维度值封装在 4 个下拉列表中，比如薪资维度的下拉列表值是：

```
<select id="salary" name="salary">
  <option value="00" selected>不限</option>
  <option value="D1" >一至三万</option>
  <option value="D2">三至五万</option>
  <option value="D3">五万以上</option>
  <option value="D8">面议</option>
  <option value="D9">其他</option>
</select>
```

如果用户不选择这个维度，在 Java 代码中做处理，则要判断 salary 的值是否是“00”，如果不是“00”，再把这个过滤条件加入过滤器列表中。在创建行键过滤器时，比较运算符使用的是 EQUAL 值，比较器类型是子串比较器。也就是说，如果用户指定的一个维度值“D8”出现在行键中，那么该行的薪资值就是“面议”类型的，该条记录就会被筛选出来。对职位名称和公司名

称的模糊检索，使用的也是同样的实现方法。对薪资维度“D8”进行过滤的步骤如图 6.8 所示。

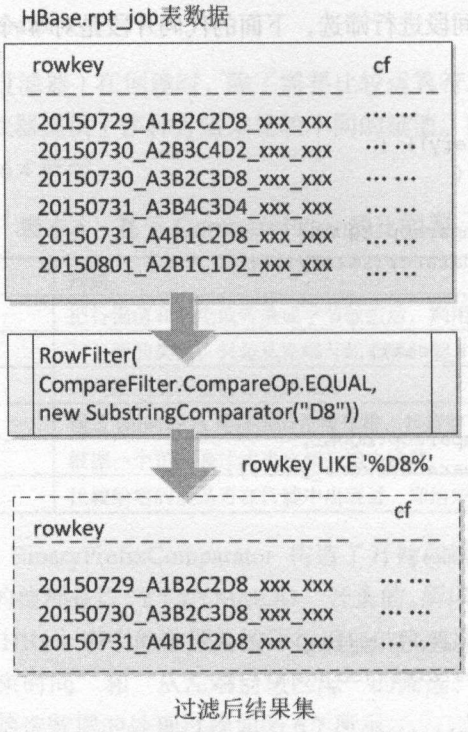


图 6.8 对薪金维度进行过滤的过程

注意：此处对于维度值和职位（公司）名称都采用了模糊匹配，但并没有考虑字符内容出现的先后次序。“20150729_A1B2C2D8_xx_xxx”中间的下画线“_”，区分出了“职位发布日期、4 个维度值、职位名称、公司名称”出现的次序。如果用户想查找有关“3D 动画设计”的职位信息时，在检索页面的“职位名称”栏目后面输入“3D”，则按照 SubstringComparator 的匹配规则，符合“YYYYMMDD_AxBxC**3D**x_xxx_xxx”模式的数据也会被检索出来，而这些数据实际上并没有对职位名称进行筛选。要解决这类问题，有兴趣的读者可以对代码做一些调整。

3. 对 HBase 返回的行值进行封装

通过使用 FilterList 把针对行键的多个 RowFilter 组装在一个列表中，下面要做的就是把这些筛选条件施加在表上，进行数据检索。代码实现很简单，如下所示：

```
HTable table=new HTable(conf, 'rpt_job');
FilterList filterList = new FilterList(FilterList.Operator.MUST_PASS_ALL);
Scan scan = new Scan();
```

```
// 添加针对行键的各种过滤条件
filterList.addFilter(new RowFilter(… …));
scan.setFilter(filterList);
ResultScanner resultScanner = table.getScanner(scan);
List<JobInfo> lst = new ArrayList<JobInfo>();

for(Result res : resultScanner) {
  lst.add(parseJobInfo(res));
}
resultScanner.close();
table.close();
```

把 filterList 赋值给 scan 对象，再把 scan 对象赋值给 HTable 实例，就实现了筛选。筛选完成返回的数据集，放在了 ResultScanner 对象中。由于数据可能存放在不同的 Region 当中，scan 扫描器在运行时并不会通过一次 RPC 调用返回所有匹配的行，而是由 ResultScanner 把每行数据封装成一个 Result 实例，把这些 Result 实例放入一个迭代器（Iterator）中返回。调用的客户端用 next() 方法遍历即可。这种处理过程，和使用 JDBC 查询数据库，将结果集封装在 ResultSet 中的做法类似。

但是，有别于 JDBC 的 ResultSet，ResultScanner 迭代器中的 Result 实例包含了一行数据的 rowkey 和所有列的值。如果使用 toString()方法打印出来，则类似于如下内容：

```
keyvalues={
20150501_01A2B2C3D8_\xE9\xAB\x98\xE7\xBA\xA7Java\xE6\x9E\xB6\xE6\x9E\x84\xE5\xB8\x8
8\xEF\xBC\x88\xE5\xA4\xA7\xE6\x95\xB0\xE6\x8D\xAEHadoop\xE6\x96\xB9\xE5\x90\x91\xEF
\xBC\x89_\xE7\xBF\x94\xE5\x82\xB2\xE4\xBF\xA1\xE6\x81\xAF\xE7\xA7\x91\xE6\x8A\x80\x
EF\xBC\x88\xE4\xB8\x8A\xE6\xB5\xB7\xEF\xBC\x89\xE6\x9C\x89\xE9\x99\x90\xE5\x85\xAC\
xE5\x8F\xB8/cf:company_address/1432733603265/Put/vlen=0/mvcc=0,
20150501_01A2B2C3D8_\xE9\xAB\x98\xE7\xBA\xA7Java\xE6\x9E\xB6\xE6\x9E\x84\xE5\xB8\x8
8\xEF\xBC\x88\xE5\xA4\xA7\xE6\x95\xB0\xE6\x8D\xAEHadoop\xE6\x96\xB9\xE5\x90\x91\xEF
\xBC\x89_\xE7\xBF\x94\xE5\x82\xB2\xE4\xBF\xA1\xE6\x81\xAF\xE7\xA7\x91\xE6\x8A\x80\x
EF\xBC\x88\xE4\xB8\x8A\xE6\xB5\xB7\xEF\xBC\x89\xE6\x9C\x89\xE9\x99\x90\xE5\x85\xAC\
xE5\x8F\xB8/cf:company_desc/1432733603265/Put/vlen=42/mvcc=0,
…… }
```

上面的代码打印出了一行数据中的 cf:company_address 和 cf:company_desc 两列的值（粗体字表示出列族名），列族名前面的内容是 rowkey，rowkey 中以“\x”开头的字符，是 UTF-8 字符的十六进制字节码表示。HBase 中的汉字采用 UTF-8 编码方式保存成字节数组的形式，每个汉字在 UTF-8 编码环境下占用 3 个字节位置。keyvalues{}可以理解成一个 list 列表，列表中的每个元素是按照“行键值/列名/列值版本号/...”方式打印出来的。

对 HBase 中的 rpt_job 表的检索，是从页面调用发起的，当然检索结果也要显示在页面中。在页面展示方面，使用了 ExtJS 技术，它要求表格中的数据行封装成 JSON 格式展示。对于上面 Result

实例得到的数据行，还必须转换成 JSON 格式。如果是 JDBC 的 ResultSet，则直接调用其中的 ResultSet.getString("列名")即可。但此处 HBase API 的 Result 对象中所有的列名和列值是混在一起的，如何与 JSON 对象中的属性匹配呢？

为了方便页面展示查询结果，我们构造了 JobInfo 实体类来封装 rpt_job 表中的每个列值。JobInfo 类定义了页面上需要展示的各个属性，以及对每个属性的 set/get 方法，代码片段如下：

```
public class JobInfo {
  private String web_id;
  private String job_url;
  private String job_name;
  private String job_location;
  private String joblocation_type;
  private String job_desc;
  private String edu;
  private String edu_type;
  ……
  ……
}
```

此外，在从 ResultScanner 迭代器中取得每行数据后，把该行数据放入自定义的列值解析方法 parseJobInfo()中进行处理。代码如下：

```
/**
 * 把 HBase 中的一行数据转换成一个 Java Bean 实例
 */
private JobInfo parseJobInfo(Result res) {
  int start = 0;
  if (res == null) {
    System.out.println("Result is null");
    return null;
  }

  JobInfo data = new JobInfo();
     for(KeyValue kv:res.list()) {
        String rowkey = new String(kv.getRow());
         if (start == 0) {
           data.setRowkey(rowkey);
         }
         else {
           data.setValue(new  String(kv.getQualifier()),new String(kv.getValue()));
         }
         start++;
     }
     return data;
}
```

该方法的输入参数是 HBase API 的 Result 对象，Result 对象中包含了所有匹配的数据行，使用 Result.list()方法可以以列表的形式得到这些数据行，列表中的每个元素都是 KeyValue 类型的。使用 getQualifier()方法可以得到所有列族名，而使用 getValue()方法可以获得该列族名对应的最新版本的列族值。

JobInfo 类中的属性名称和 rpt_job 表中的列族名是一致的。由于我们需要把 rpt_job 表中的所有列族值自动赋值给 JobInfo 类的属性，因此在 JobInfo 类中定义了 setValue(String cqName, String cqValue) 方法。该方法的代码如下：

```
/**
 * 从 HBase 的列族中得到列名和列值，赋值给对应的 Bean 属性
 * @param name - 列名
 * @param value - 列值
 */
private void setValue(String name, String value) {
  if ("company_name".equals(name)) {
    this.company_name = value;
  }
  if ("job_name".equals(name)) {
    this.job_name = value;
  }
  if ("job_location".equals(name)) {
    this.job_location = value;
  }
  ......
  ......
}
```

rpt_job 表中的每个列族名都对应于 JobInfo 类的属性名，上面的代码就是依次把列族值赋值给属性值。如果读者觉得这种写法太笨拙（如果有几十个属性要怎么写），则可以使用 Java 反射功能来自动匹配属性名并赋值。

从 HBase 中查询出来的数据行，最后是要显示在页面上的。页面展示采用了 ExtJS 技术做简单处理，HBase API 查询和页面之间的控制器层可以采用多种实现方式，比如 Servlet/Struts Action/Spring MVC 等。这些基本的 Java Web 技术不在本书讨论范围内，故此处不做详细介绍。图 6.9 展示了前端页面和 HBase 数据的交互过程。

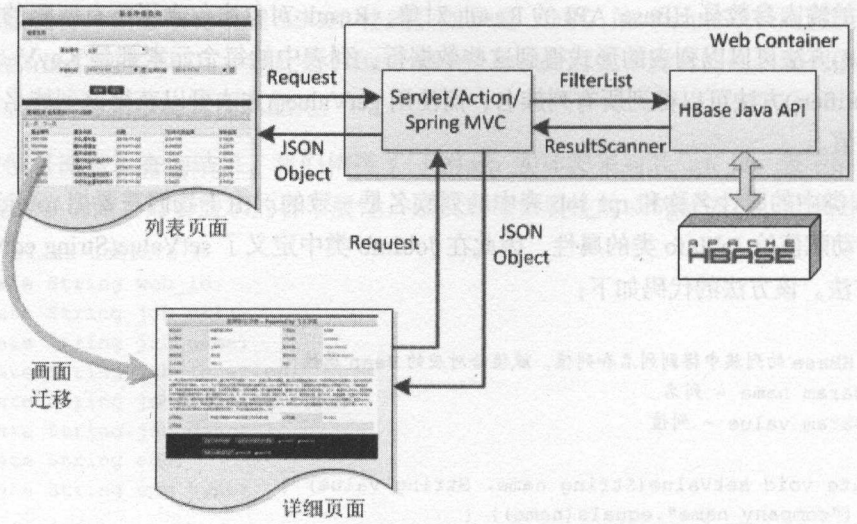


图 6.9 前端页面和 HBase 数据交互过程

6.6 如何显示职位表中的某条具体信息

用户在页面上输入检索条件，查询结果以列表形式返回，只显示职位的概要信息（包括行键值）。如果想查看职位的详细信息，就要再次对 HBase 的 rpt_job 表进行检索。这次检索是在已知行键值的情况下进行的，HBase API 中提供了直接根据 rowkey 值检索的方法—— HTable.get(String rowkey)。示例代码如下：

```
public JobInfo getJobInfoByRowkey(String rowkey) throws IOException {
    // 创建一个 HBase 配置实例
    Configuration conf = HBaseConfiguration.create();
    conf.set("hbase.zookeeper.quorum", zkQuorum);
    conf.set("hbase.zookeeper.property.clientPort", zkClientPort);
    conf.set("hbase.master", zkMaster);
    JobInfo jobInfo = new JobInfo();
    HTable table = new HTable(conf, "rpt_job");

    // 使用 get 方法从 table 中根据行键值得到所有列族值
    Get get = new Get(Bytes.toBytes(rowkey));
    Result result = table.get(get);
    Cell[] cells = result.rawCells();

    // 把所有列族值构造成一个对象返回
    CommonUtils.convertCell2JavaBean(jobInfo,cells);
    table.close();
```

```
    return jobInfo;
    }
```

把所有列族值构造成一个对象返回，示例代码如下：

```
public static void convertCell2JavaBean(Object objInstance,Cell cells[]) {
    try {
        Class<? extends Object> classType = objInstance.getClass();
        Field fields[] = classType.getDeclaredFields();
        for(Cell c:cells){
            String qualifier = Bytes.toString(CellUtil.cloneQualifier(c));
            String value = Bytes.toString(CellUtil.cloneValue(c));
            System.out.println("列族名:" + qualifier);
            System.out.println("列族值:" + value);
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

至此，查询页面和职位明细页面都已经实现了 HBase 数据表的实时检索功能。

6.7　本章小结

HBase 是 Hadoop 生态系统中 NoSQL 思想的具体实现者。本章对 HBase 中的几个重要概念和常用操作做了简要介绍。以完成飞谷项目需求为主要目的，讲解了如何设计职位表的行键，如何把 Hive 中的数据导入到 HBase 中，以及如何使用 HBase Java API 完成数据检索。通过实现过程的描述，使读者对 HBase 的用法有了大致的了解。

目前国内很多大型互联网公司中，HBase 的使用非常多。围绕 HBase 展开的集群搭建管理、性能监控、故障调试，甚至基于源码的二次开发都有深层次的实践。希望通过本章的讲解，读者能够围绕 HBase 多找出一些“为什么”，带着疑问去翻阅更多的 HBase 书籍资料。

第 7 章
大数据的展示

7.1 概述

存放在 Hadoop 系统中的数据，如何展示给终端用户？展示时是使用文字描述还是其他方式？目前市场上流行有哪些工具？这是本章要讨论的问题。

2014 年 1 月下旬，央视在晚间新闻栏目推出的“‘据’说春运”特别节目，首次采用百度地图方式来展示春运期间人口迁徙情况，相信给很多观众都留下了深刻印象（见图）。这是大数据可视化方面的一个典型案例——原来数据展示还可以这样做！样本数据来自百度地图的开放平台，每个使用百度定位功能的移动终端都会被纳入统计样本中。另外，统计时间窗固定在 8 小时内，样本数量有上亿个之多，同一个终端在两个不同地点出现就连上一条线，这样做出来的全量分析结合地图展示出来。

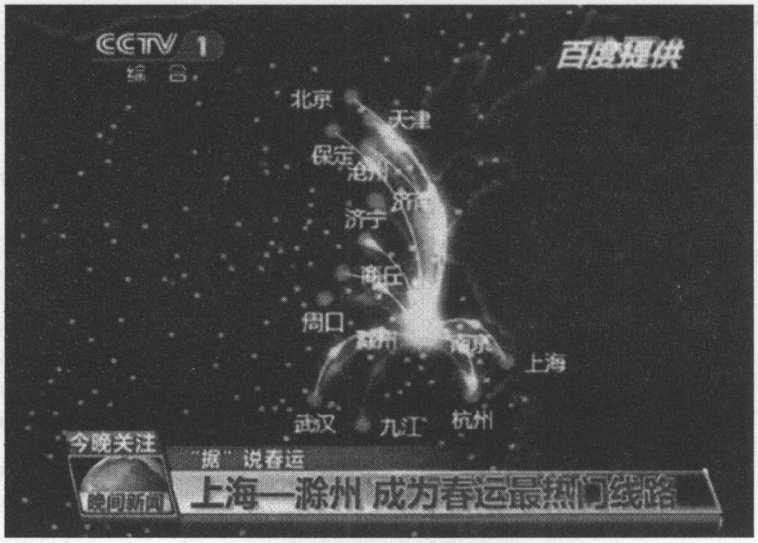


图 7.1 “据”说春运（图片来源于央视晚间新闻画面）

数据展示方式有多种，如文字、表格、图形、音频、视频等均可。文不如表，表不如图，一图胜千言，已是共识。而数据可视化技术就是对数据视觉表现方式的研究，是将大型数据集中以图形图像等直观形式表现出来，便于快速、准确地传达信息。并且，可以利用数据分析和开发工具发现其中有价值的信息，为决策提供参考。

在大数据时代，数据分析最终要通过数据可视化来进行呈现，可视化工具多种多样，有 Excel 图表、IBM 的 SPSS 和 Cognos、SAS 等传统的商业智能和数据挖掘分析工具，也有近些年很热门的 R 和 Tableau，还有各种层出不穷的基于 Web 浏览器端的 JavaScript 图表库，比如百度 ECharts、Google Chart API、D3 等。每种工具支持的数据来源类型、功能强弱、适用场景及学习曲线均不同。本章讲解如何结合使用 R 语言来完成数据展示。

7.2　数据分析的一般步骤

虽然分析工具有很多，但进行数据分析的步骤大体上是相同的，即都经过数据获取（整理）、分析、呈现三个环节。比如以大家都很熟悉的 Excel 电子表格为例，我们想实现一年内每月销售量的占比图，首先就要把一年的所有订单都录入到 Excel 里面，这叫作数据获取；然后利用 Excel 的函数功能进行按月分组、对销量求和，完成数据分析；最后选择柱状图或饼图进行数据呈现。

一个好用的数据展示工具，在这三个方面都是很优秀的。除此之外，能适用多种场合，且学习起来容易上手，也是很重要的衡量标准。接下来通过一个实际的例子，讲解在飞谷项目中如何分步骤进行数据分析展示。

7.3　用 R 来做数据分析展示

在飞谷项目中，使用 R 制图系统来完成数据展示环节。R 是一套用于统计分析、制图的软件系统和开发环境，是属于 GNU 范畴的免费、开源软件，官网地址是 www.r-project.org。R 以其突出的数据分析能力和强大的工具包支持，得到了很多专业统计人员的青睐。由于 R 具有免费、开源的特点，因此始终有很多的追随者。飞谷项目之所以采用 R 工具，也是看中了其免费、跨平台的优势。并且，Hadoop 大数据加上 R 语言展现，这种组合无论从技术实现还是成本上，都很适合科研院校及初创企业做架构选型使用。本章的例子就来展示如何以 Hadoop HDFS 中的数据作为数据源，采用 R 语言生成统计图表。

7.3.1　在 Ubuntu 上安装 R

在 Linux 系列操作系统上安装 R 环境，通常有两种方式：一是在线安装，使用“sudo apt-get install r-base”或“yum”命令，在本机上直接安装；二是下载某个特定版本的 R 源码包，手动编译安装。

本节采用手动编译方式，选择 R 的 3.1.3 版本（使用此版本是为了能和 Spark 整合在一起使用）。下面是在 Ubuntu 上安装的关键步骤。

从 CRAN 官网（cran.r-project.org）下载 R 源码包，选择 3.1.3 版本，存放到本地后，解压缩 tar 文件。

在编译 R 源码前，要先保证 Java 环境已经正确安装，并能访问到环境变量“JAVA_HOME”，如下所示。此处略去 JDK 安装步骤。

```
[open@demo ~]$ java -version
java version "1.8.0"
Java(TM) SE Runtime Environment (build 1.8.0-b132)
Java HotSpot(TM) 64-Bit Server VM (build 25.0-b70, mixed mode)
[open@demo ~]$ echo $JAVA_HOME
/home/open/jdk1.8.0
```

由于 R 编译时要依赖很多系统包，可以使用“sudo apt-get install <包名>”命令预先安装好这些系统包，否则 R 编译过程可能会出错。下面列出了需要安装的系统包：

- build-essential
- fort77
- xorg-dev
- liblzma-dev、libblas-dev、gfortran
- gcc-multilib
- gobjc++
- aptitude
- libreadline6-dev
- tcl-dev、tk-dev
- libicu-dev
- libpng12-dev、libjpeg-dev、libtiff4-dev、libcairo2-dev

上面最后一行的 4 个 lib 包，是为了使 R 支持图形和生成 PDF 文件。

将 tar 文件解压缩到/home/open 目录下，使用“./configure”命令配置编译环境，其中“/home/open/bigdata/R-3.1.1”是指定编译后 R 文件的安装目录；“enable-R-shlib”参数可以保证系统 lib 目录下的动态库能够被 R 共享。

```
[open@demo ~/R-3.1.3]$ pwd
/home/open/R-3.1.3
[open@demo ~/R-3.1.3]$ ./configure --prefix=/home/open/bigdata/R-3.
1.1 --enable-R-shlib  --with-x=yes
```

“./configure”命令运行完毕后，可能会出现的 R 安装环境清单内容如下：

```
Source directory:          .
Installation directory:    /home/open/bigdata/R-3.1.1

C compiler:                gcc -std=gnu99  -g -O2
Fortran 77 compiler:       gfortran  -g -O2

C++ compiler:              g++  -g -O2
C++ 11 compiler:           g++  -std=c++11 -g -O2
Fortran 90/95 compiler:    gfortran -g -O2
Obj-C compiler:            gcc -g -O2 -fobjc-exceptions

Interfaces supported:      X11, tcltk
External libraries:        readline, lzma
Additional capabilities:   PNG, JPEG, TIFF, NLS, cairo, ICU
Options enabled:           shared R library, shared BLAS, R profiling

Capabilities skipped:
Options not enabled:       memory profiling

Recommended packages:      yes
```

然后，使用“make && make install”命令编译、安装 R 环境。

```
[open@demo ~/R-3.1.3]$ make && make install
```

安装完毕后，需要设定系统（或用户）的环境变量，新增“R_HOME”以及在 PATH 变量中增加 R/bin 目录。

```
export R_HOME="/home/open/bigdata/R-3.1.1"
export PATH="/home/open/bigdata/R-3.1.1/bin:"$PATH
```

最后，确认 R 的安装版本号。

```
[open@demo ~]$ R --version
WARNING: ignoring environment value of R_HOME
R version 3.1.3 (2015-03-09) -- "Smooth Sidewalk"
Copyright (C) 2015 The R Foundation for Statistical Computing
Platform: x86_64-unknown-linux-gnu (64-bit)
```

7.3.2　R 的基本使用方式

从控制台进入 R 环境，只需要输入“R <回车>”即可。由于接下来要使用 R 生成统计图形，所以要确保 R 环境有生成 JPEG/PNG 文件的功能。

进入 R 界面，输入“capabilities()”命令，可以看到在当前环境下 jpeg 和 png 显示的均为 TRUE。

```
> capabilities()
      jpeg         png        tiff       tcltk         X11         aqua
      TRUE        TRUE        TRUE        TRUE       FALSE       FALSE
  http/ftp     sockets      libxml        fifo      cledit      iconv
      TRUE        TRUE        TRUE        TRUE        TRUE        TRUE
       NLS     profmem       cairo         ICU long.double
      TRUE       FALSE        TRUE        TRUE        TRUE
```

如果读者使用的 R 环境在编译前未安装 libpng/libjpeg 等包，要想使 R 支持 JPEG 和 X11 图形方式，可以直接安装 Cairo 组件（http://cairographics.org/），它是一个专门实现图形绘制渲染的组件，提供了一套 API，也可以在 R 环境中使用。

在安装 Cairo 组件前，首先要使用“sudo apt-get install libcairo2-dev”命令安装 cairo2 开发库，然后在 R 控制台安装 Cairo。

```
> install.packages("Cairo")
--- Please select a CRAN mirror for use in this session ---
CRAN mirror
```

选择一个镜像网站，正确安装完毕后，会出现如下提示信息：

```
** R
** preparing package for lazy loading
** help
*** installing help indices
** building package indices
** testing if installed package can be loaded
* DONE (Cairo)

The downloaded source packages are in
        ‘ /tmp/RtmproIdBd/downloaded_packages’
Updating HTML index of packages in '.Library'
Making 'packages.html' ... done
```

安装完毕后，可以加载 Cairo 库，并且可以查看到在 Cairo 中均支持创建 png/tiff/pdf 等文件类型。

```
> Cairo.capabilities()
   png   jpeg   tiff    pdf    svg     ps    x11    win raster
  TRUE   TRUE   TRUE   TRUE   TRUE   TRUE   TRUE  FALSE   TRUE
```

下面所示的代码用来测试使用 Cairo 组件生成散点图功能。

```
> library(Cairo)
> x<-rnorm(200)
> y<-rnorm(200)
> CairoPNG(file="CairoScatter.png",width=320,height=240)
> plot(x,y,col="#6600CC", lty=1, lwd=1, pch=17,cex=2,main = "ca
iro scatter demo")
> dev.off()
null device
          1
```

生成的散点图如图 7.2 所示。

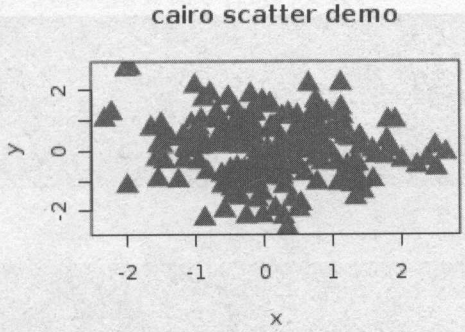


图 7.2　Cairo 生成的散点图

7.4　用 Hive 充当 R 的数据来源

使用 R 作为数据挖掘和展示的工具，其支持的数据来源是多种多样的。在 R 中可以使用爬虫技术从网上抓取数据来进行分析，也可以从平面文件 txt/csv 中读取数据，还可以从 RDBMS 中读取数据表信息。本节就来讲述如何从 HDFS 中读取数据，并最终把统计图表展现在 Java Web 查询页面中。

7.4.1　RHive 组件

对于 R 来讲，Hive 只是一种数据源类型。就像 Java 使用 JDBC 封装的接口来读取不同的数据源一样，RHive 封装了把数据从 Hive 读取到 R 中的具体实现逻辑，而 R 开发人员不必关心数据加载过程，只需要把 RHive 包引入到 library 中，然后通过 HQL 读取 Hive 中的数据，转换成 R 中的集合对象来调用即可。

1. RHive 组件的安装

在 R 界面操作 Hive 中的数据，主要是通过向 HiveServer 发送 HQL 语句的方式来执行的。在 5.9.1 节中提到过，操作 Hive 表中的数据可以通过 Hive CLI 方式或 beeline 方式。RHive 的作用和 beeline 类似，它把一个操作 Hive 表数据的客户端工具嵌在了 R 中。

RHive 组件的运行需要 Java 环境，所以要先安装 rJava 组件，再安装 RHive，然后启动 HiveServer/HiveServer2，最后在 rhive 对象中通过执行 HQL 语句操作 Hive 表中的数据。

（1）安装 rJava 组件

选择一个镜像网站，下载并安装 rJava 组件。

```
> install.packages("rJava")
Installing package into ' /home/open/R/x86_64-unknown-linux-gnu-li
brary/3.1'
(as ' lib'  is unspecified)
--- Please select a CRAN mirror for use in this session ---
CRAN mirror
```

测试 rJava 模块是否可以正常使用。

```
> library(rJava)
> .jinit()
> s <- .jnew("java/lang/String", "Hello World")
> s
[1] "Java-Object{Hello World}"
>
> search()
 [1] ".GlobalEnv"        "package:rJava"     "package:stats"
 [4] "package:graphics"  "package:grDevices" "package:utils"
 [7] "package:datasets"  "package:methods"   "Autoloads"
[10] "package:base"
>
```

（2）安装 RHive 组件

选择一个镜像网站，安装 RHive 组件。

```
> install.packages("RHive")
Installing package into ' /home/open/R/x86_64-unknown-linux-gnu-lib
rary/3.1'
(as ' lib'  is unspecified)
--- Please select a CRAN mirror for use in this session ---
CRAN mirror
```

安装完毕后，测试 RHive 模块是否正常加载。

```
> library(RHive)
> search()
 [1] ".GlobalEnv"        "package:RHive"     "package:rJava"
 [4] "package:stats"     "package:graphics"  "package:grDevices"
 [7] "package:utils"     "package:datasets"  "package:methods"
[10] "Autoloads"         "package:base"
>
```

在使用 RHive 组件功能前，必须先启动 HiveServer 服务端。

安装 RHive 组件，除了使用函数 install.packages("RHive")之外，还可以通过命令行方式下载安装 RHive 开发包。用户可以先到 http://cran.at.r-project.org/src/contrib/Archive/RHive 地址，下载 RHive_2.0-0.2.tar.gz 压缩包，然后利用 R 命令行进行安装。

```
feigu@slave01:~$ R CMD INSTALL RHive_2.0-0.2.tar.gz
WARNING: ignoring environment value of R_HOME
* installing to library ‘ /home/feigu/R/x86_64-unknown-linux-gnu-library/3.1’
* installing *source* package ‘ RHive’  ...
** package ‘ RHive’  successfully unpacked and MD5 sums checked
** R
** inst
** preparing package for lazy loading
** help
*** installing help indices
** building package indices
** testing if installed package can be loaded
* DONE (RHive)
feigu@slave01:~$
```

默认 RHive 安装包的位置在~/R/x86_64-unknown-linux-gnu-library/3.1/RHive 下，如上面安装信息中所示。然后需要把该目录下的 java/rhive_udf.jar 文件复制到 HDFS 文件系统中的 /rhive/lib/2.0-0.2 目录下，在 HDFS 中目录可以先手动创建好。

（3）启动 HiveServer 服务端

该步骤可参考“5.9.2 Hive Thrift 服务”章节内容。

（4）在 R 中使用 RHive 连接 Hive 数据仓库

本例中，在本机上启动了 Hive Thrift 服务，因此连接地址是“localhost”。

```
> library(RHive)
Loading required package: rJava
> rhive.init()
> rhive.connect('localhost')
2015-10-18 22:54:48,370 INFO  Configuration.deprecation (Configuration.java:
warnOnceIfDeprecated(1019)) - fs.default.name is deprecated. Instead, use fs
.defaultFS
SLF4J: Class path contains multiple SLF4J bindings.
```

connect 命令中的参数，还有一种语法形式是 rhive.connect('localhost',11000,hiveServer2=TRUE, defaultFS="hdfs://master:9000")，其中第一个参数是要连接的服务器地址；第二个参数是服务端口号，

第三个参数是是否默认使用 HiveServer2 服务类型；第四个参数是指定默认的 HDFS 文件系统位置。

（5）利用 HQL 语句操作 Hive 表中的数据

```
> rhive.query('use feigu3')
Error: java.sql.SQLException: The query did not generate a result set!
> rhive.query('show tables')
                tab_name
1          daily_dim_sum
2                dim_edu
3       dim_job_techword
4        dim_joblocation
5   dim_joblocation_test
6      dim_news_techword
7    dim_resume_techword
8             dim_salary
9           dim_workyear
10               dm_job
11          dm_job_test
12               rpt_job
13          rpt_job_test
14                 s_job
15               stg_job
16              stg_news
17            stg_resume
18                  test
```

（6）退出 RHive 时，要关闭数据库连接

```
> rhive.close()
[1] TRUE
>
```

2. 从 Hive 职位表中生成饼图

在“5.3.1 逻辑模型的创建”章节中，每日维度统计表（daily_dim_sum）用来存放 4 个统计维度每日的单项汇总信息，比如“20150501”日期分区下的数据为：

dim_type	cnt_val
A1	8
A2	1
A3	1
A4	2
A9	5
B1	1
B2	9
B3	1
B9	6
D1	3
D8	14
C1	2
C2	5
C3	4
C9	6

在前端职位统计图表页面中，用户可以选择任意一个维度（学历、工作年限、职位地域、薪资），以饼图方式查看在一定时间段内该维度中的各个取值占比。如图 7.3 所示是生成的 2015 年 6 月工作经验分布图效果。

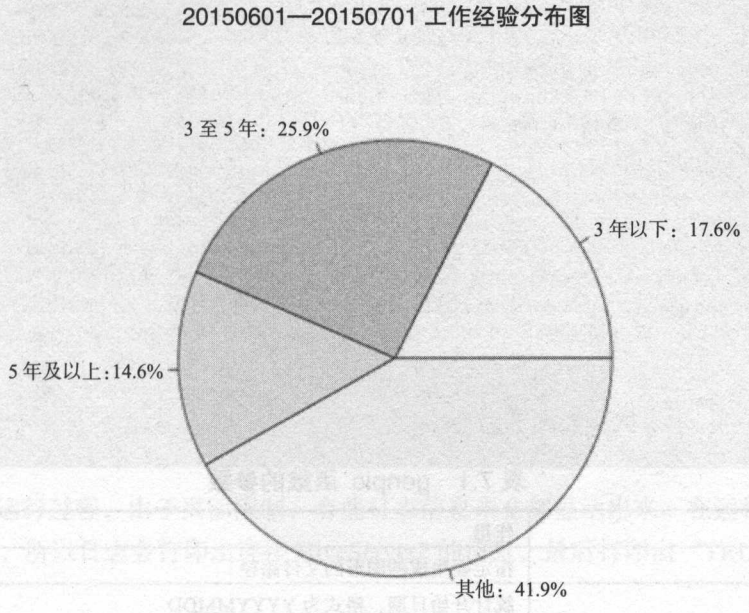


图 7.3　统计图形

对于以上图形，数据是从 Hive 取得的，图形是用 R 生成的，生成图形的过程封装在一个 R 自定义函数 genpic 中，函数代码如下：

```
genpic<-function(picpath,dat1,dat,dimtype){

library(rJava)
library(RHive)
rhive.init()
rhive.connect("slave02",10001,hiveServer2=TRUE,defaultFS="hdfs://master:9000")
try(rhive.query('use feigu3'),silent=TRUE)

a<-rhive.query(paste('select * from daily_dim_sum where daily_dim_sum.pt between ',dat1,
'and',dat,sep=" "))
b<-aggregate(a$daily_dim_sum.cnt_val,by=list(a$daily_dim_sum.dim_type),FUN=sum)
b<-b[order(b[,1]),];

xx<-data.frame(x=c(paste('A',c(1:4,9),sep=""),paste('B',c(1:3,9),sep=""),paste('C',
c(1:3,9),sep=""),paste('D',c(1:3,8:9),sep="")), y=c('北京','上海','广州','深圳','其他
','大专','本科','研究生','其他','3 年以下','3 至 5 年','5 年及以上','其他','一至三万','三至五万
```

```
','五万以上','面议','其他'));
xx$y<-as.character(xx$y);

b$region<-xx$y[xx$x %in% b[,1]];

  ##output.filepath 图形输出路径, title 图片标题行
  colnames(b)[2]<-'amount';       ##设置列名
  b$dim1<-substr(as.character(b[,1]),1,1);
  b$ratio<-round(with(b,b$amount*4/sum(b$amount)),3)  ##计算百分比
  title<-c('地域','学历','工作经验','月薪')
  na<-paste(picpath,1:4,'.jpeg',sep="")
i<-dimtype;
jpeg(na[i],family='GB1');
pie(b$ratio[b$dim1==unique(b$dim1)[i]],labels=paste(b$region[b$dim1==unique(b$dim1)
[i]],":",b$ratio[b$dim1==unique(b$dim1)[i]]*100,"%",sep=""),family='STKaiti')##画饼图
title(main=paste(dat1,'-',dat,title[i],'分布图',sep=""),family='STXihei')##设置图片 title
dev.off()          ##关闭画图
}
```

genpic 函数有 4 个参数，如表 7.1 所示。

表 7.1　genpic 函数的参数

参数名	作用
pic_path	指定要生成的图形的文件路径
start_dt	统计开始日期，格式为 YYYYMMDD
end_dt	统计结束日期，格式同上
dim_type	维度类型（1、2、3、4 分别代表学历、工作年限、职位地域、薪资 4 种）

此外，对于以上代码要注意如下几点。

- 首先在函数开始部分，将 rJava 和 RHive 加载到 library 中，创建出 rhive 对象，并进行初始化。rhive.connect('localhost')是连接本机的 HiveServer2 服务，如果 HiveServer2 服务在其他机器上启动，则要把 localhost 换成该机器的主机名或 IP 地址。注意：rhive 对象在使用完毕后，一定要调用 rhive.close()方法关闭数据库连接。
- 向 Hive 中发送 HQL 是通过调用 rhive.query()方法完成的。获取 daily_dim_sum 表中的数据，使用 select 语句实现，得到的结果集放入 R 的 dataframe 对象中进行后续处理。
- 生成的 JPEG 图形，包括标题 title 和饼图两部分，在示例代码中使用了楷体字体 STKaiti，可以用其他的中文字体替代。
- 生成的 JPEG 图形名称，由输入的参数拼接构成，比如调用参数为 genpic('/home/feigu/rpic/','20150601','20150630',2)，则生成的图形位置及名称为“/home/feigu/rpic/20150601-20150630-2.jpeg”。

在 R 终端运行 genpic 脚本示例如下：

```
> source('/home/feigu/tomcat6/webapps/feigu-web/WEB-INF/RHiveDatesCairo.R')
> genpic("/home/feigu/tomcat6/webapps/feigu-web/rimg/", "20150901", "20150917", 4)
2015-09-27 16:34:32,255 INFO  Configuration.deprecation (Configuration.java:warnOnceIfDep
 Instead, use fs.defaultFS
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/hadoop/bigdata/hadoop/share/hadoop/common/lib/slf
erBinder.class]
SLF4J: Found binding in [jar:file:/home/hadoop/bigdata/hadoop/share/hadoop/httpfs/tomcat/
.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
2015-09-27 16:34:32,815 WARN  util.NativeCodeLoader (NativeCodeLoader.java:<clinit>(62))
 platform... using builtin-java classes where applicable
Warning:
        +------------------------------------------------------------+
        + / hiveServer2 argument has not been provided correctly.    +
        + / RHive will use a default value: hiveServer2=TRUE.        +
        +------------------------------------------------------------+

[1] TRUE
>
```

上面截屏的运行过程，由于页面限制，有些日志信息未全部显示出来。在运行的 R 脚本中使用了 RHive 模块，所以日志会打印出连接 HiveServer2 的信息，最后打印出“TRUE”表示图形已经生成。

生成的图形位于 genpic 的第一个参数指定的目录下，如下所示：

```
open@slave01:~$ ls  /home/feigu/tomcat6/webapps/feigu-web/rimg/20150901-20150917-4.jpeg
/home/feigu/tomcat6/webapps/feigu-web/rimg/20150901-20150917-4.jpeg
```

需要注意的是，进入 R 终端时所使用的用户（比如 feigu），应该对图形生成目录有写权限，否则会提示权限错误。另外，如果在启动 HiveThriftServer/HiveServer2 服务时，用 root 用户启动，而用 feigu 用户启动 R 客户端，使用 RHive 组件连接 HiveServer，也会出现权限错误提示。

7.4.2　把 R 图表整合到 Web 页面中

前面所生成的统计图形，是通过在 R 控制台输入命令行实现的。在 Web 检索页面中要实现的功能是，用户点击“检索”后，在页面下方直接显示所生成的图形，如图 7.4 所示。

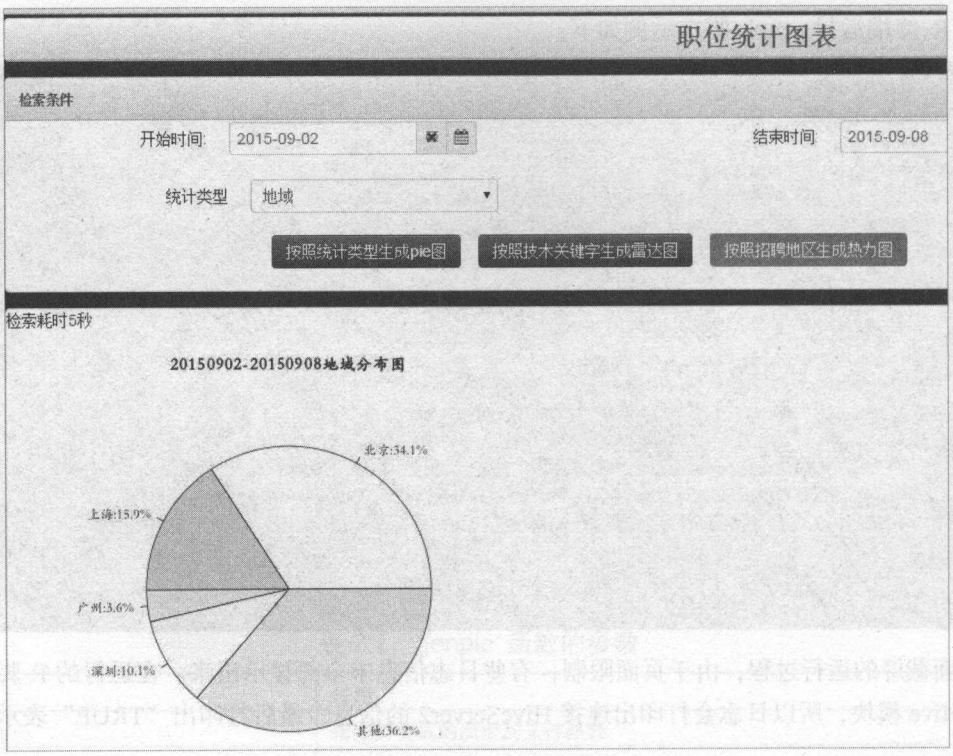


图 7.4　职位统计 Pie 图

Web 部分的功能都是用 Java/JSP 技术实现的，那么有没有一种整合 R 和 Java 的方案呢？

1. 用 Java 调用 R 脚本

Java 语言和 R 语言的运行环境及方式完全不同，如何实现相互调用呢？

异构语言之间的相互调用，即进程间通信，要解决两个关键问题：一是如何通知对方；二是如何把参数或返回值传递给对方。不同语言的语法结构和变量类型定义均不相同，那么如何把参数做到标准化呢？最简单的方式是使用字符串。无论任何语言，都有基本的字符串类型，把数字也转换成字符串格式，将参数变量保存到文件、数据库或共享存储中。两种不同语言的内存进程，采用“轮询”方式，一方写数据，一方读数据。这是最简单的实现方式。

考虑到“轮询”方式耗费资源，且实时性不强，很多高级语言都支持 Socket 通信，因此使用 Socket Server/Client 方式，也是非常通用的方法。把需要传递的参数，以 Socket 报文方式发送到服务器端，服务器端执行完成后，再把返回值或结果集以报文方式发送到调用端。

由于 Socket 通信报文长度会有限制，无法适用于更加复杂的场合，因此各种语言又发挥各自

所长，提供了 Web Service 方式实现 RPC 调用，这也是跨语言的通信。

此处要实现由 Java 端发起调用 R 的服务，我们采取一种和上述方案都不同的方法，该方法既直观又容易实现。

在 R 中生成饼图的 genpic 函数，可以封装成一个 R 脚本。所谓 R 脚本（R Script）是指包含了多行 R 命令的一个文本文件，每行 R 命令在 R 终端都可以直接运行。把 R 命令封装成 R 脚本文件是为了方便在不同地方重复使用，就像把很多 JavaScript 代码放到一个*.js 文件中，在 HTML 页面中直接引用一样。执行 R 脚本文件不需要登录到 R 控制台，直接在操作系统终端运行即可。

在机器上安装好 R 环境后，在${R_HOME}\bin 目录下会有 Rscript.sh（或 Rscript.exe）文件，通过执行 Rscript 命令可以在 OS 终端直接运行 R 脚本文件。运行方式为：

```
Rscript <文件路径/R 脚本文件名>  [R 脚本中参数列表]  <回车>
```

首先把 genpic 函数放在 R 脚本文件中，命名为 genpicScript.R。扩展名一般为.R，文件格式是文本文件，字符集采用 UTF-8。文件内容大致如下：

```
#从操作系统命令行得到参数列表，放入 Args 数组中
Args<-commandArgs()
#定义 genpic 函数
genpic<-function(picpath,dat1,dat,dimtype){
  #以下省略函数内容
}

#调用 genpic 函数，参数是从控制台获取的
genpic(Args[1],Args[2],Args[3],Args[4])
```

把 R 脚本文件保存在/home/feigu/rscript/genpicScript.R 路径下，在 Windows 或 Linux 终端输入命令行调用。

```
Rscript /home/feigu/rscript/genpicScript.R
'/home/feigu/rpic/'  '20150601'  '20150630'  2
```

脚本文件名后的参数之间用空格分隔，参数值与在 R 客户端调用的含义相同。通过运行 R 脚本文件，同样可以生成图形文件。

下面轮到 Java 出场了。在 Java 的 Runtime 类中，可以模拟在操作系统终端执行一个系统命令或外部命令的功能。也就是说，刚才我们用 Rscript 运行 R 脚本文件的操作，可以移到 Java 代码中实现。代码如下：

```
try {
    String cmd = "Rscript /home/feigu/rscript/genpicScript.R "+file_path+" "+date;
    Runtime.getRuntime().exec(cmd);
    System.out.println("Done");
} catch (IOException e) {
```

```
        e.printStackTrace();
    }
```

把要执行的命令行作为一个字符串，放入 Runtime 类的 exec 方法参数中，就可以实现执行命令行的效果，也就间接地实现了 Java 调用 R 脚本生成图形的功能。调用 R 脚本后，在 Java 代码中判断指定目录下的 JPEG 文件是否生成，如果生成了，就将文件路径返回给前端 Web 页面显示。

使用 Rscript 命令行执行 R 脚本，可以把复杂的实现逻辑封装在脚本文件中，先在 R 终端调试脚本内容，确保语法无误后再使用 Java 代码来调用，这样就大大降低了系统的耦合度，也方便进行错误调试。

使用 Runtime.exec()方法执行外部命令的方式，可以实现 Java 和其他各种语言进行交互的功能。比如 PHP、Python、Wscript 等脚本语言，均提供了在命令行执行的功能：“php listFile.php”、“python myFile.py”、“wscript myScript.vbs”（vbs 文件是用 VBScript 语言编写的宏指令集合）。

但是，使用 Runtime.exec()也有一些限制条件。最大的缺点是，运行 exec()参数里面的命令行，要求本机必须安装了执行命令行的软件环境。比如用户在 A 机器上运行 Java 代码 Runtime.getRuntime.exec("Rscript drawPie.R")，要求 A 机器上必须已经安装了 R 环境，否则会报错“找不到 Rscript 命令”，其他命令行同理。

此外，使用 Runtime.exec("系统命令行")方式，在运行命令行时，操作系统另外启动了一个进程来执行“系统命令行”，Java 进程本身还是继续向下运行，两个进程是异步方式。如果系统命令需要运行一段时间，而此时 Java 进程已经结束或返回，这也不是我们所期望的。

2. 用 Rserve 实现实时通信

用 Java 调用 R 脚本更高效的方法是使用 Rserve 通信方式。Rserve 组件是 R 提供的采用 TCP/IP 协议、以监听方式执行 R 命令的服务端组件。客户端调用时可以采用其他语言，比如 C/C++、PHP、Java、Python、C#等，客户端机器并不需要安装 R 环境，类似于 Hive 的 ThriftServer。

Java 在调用 Rserve 服务时需要两个 JAR 包支持：REngine.jar 和 RserveEngine.jar，可以从 http://www.rforge.net/Rserve/files/下载得到。Java 和 Rserve 之间的调用步骤如下。

（1）安装 Rserve 模块

进入 R 终端，输入 install 命令安装，如下所示，选择一个镜像地址。

```
> install.packages("Rserve")
Installing package into ‘ /home/open/R/x86_64-unknown-linux-gnu-library/3.1’
(as ‘ lib’  is unspecified)
--- Please select a CRAN mirror for use in this session ---
CRAN mirror
```

（2）在 R 终端启动 Rserve

安装过程很快，安装完毕后可以直接启动 Rserve。

```
> library("Rserve")
> Rserve()
Starting Rserve:
 /home/hadoop/bigdata/R-3.1.3/bin/R CMD /home/open/R/x86_64-unknown-l
```

也可以在控制台输入“R CMD Rserve”来启动 Rserve。Rserve 启动后，提供服务的默认端口号是 6311，可以输入“R CMD Rserve --help”命令来查看如何修改默认端口号。

Rserve 启动后，默认只允许从本机发起连接。如果想将其他主机作为客户端连接 Rserve，则需要打开远程连接模式，启动命令是“R CMD Rserve--RS-enable-remote”。关于 Rserve 的详细配置信息，可以查看 http://www.rforge.net/Rserve/doc.html。

（3）在 Java 代码中通过端口号连接 Rserve

Rserve 服务启动后，可以编写测试代码连接 Rserve 做测试。

```
import org.rosuda.REngine.REXP;
import org.rosuda.REngine.Rserve.RConnection;
.....
.....
public static  void callRScript()throws RserveException, REXPMismatchException {
    String rserver_ip = "127.0.0.1";
    int rserver_port = 6311;
    RConnection  conn = new RConnection(rserver_ip, rserver_port);
    System.out.println("connection ok..");
    conn.close();
}
```

此处是在本机上启动 Rserve 守护进程的，Java 代码连接的是本机 IP 地址 127.0.0.1。IP 地址也可以换成其他启动 Rserve 服务的机器地址。

（4）以字符串方式发送 R 命令并执行

```
RConnection conn = new RConnection(rserver,rserver_port);
REXP rexp = conn.eval("R.version.string");
System.out.println(rexp.asString());// 打印 R 版本号

double[] arr = conn.eval("rnorm(10)").asDoubles();
for (double a : arr) {// 循环打印变量 arr 中的每个随机数
    System.out.print(a + ",");
}
conn.close();
```

RConnection 类使用 eval("R 命令行")方法来解析执行参数中的 R 命令，并把返回值放入对象

REXP 实例中。此处执行的 R 命令行只是生成图形，并不需要解析复杂的返回值。有关 eval 方法的详细使用说明，可以参考 http://www.rforge.net/doc/packages/Rserve/Rserve.eval.html。

（5）引入 genpicScript.R 脚本文件，并执行其中的 genpic 函数生成图形

```
public static boolean callRScript() {
    String rserver_ip = "127.0.0.1";
    int rserver_port   = 6311;
    String rscript_file = "/home/feigu/genpicScript.R";
    String output_filepath = "/home/feigu/rimg/";
    String start_dt = "20150601";
    String end_dt = "20150630";
    String dim_type = "2";
    RConnection conn = null;
    try {
        // 创建 RServer 连接
        conn = new RConnection(rserver_ip rserver_port);
        System.out.println("connection ok..");
        // 加载 R 脚本文件，里面定义了 genpic 函数
conn.eval("source('"+rscript_filelocation+"')");
        System.out.println("load source Rfile ok..");
        // 调用 genpic 函数生成饼图
        REXP x = conn.eval(String.format("genpic(\"%s\", \"%s\", \"%s\",%s)", output_
filepath,start_dt,end_dt,dim_type));
        System.out.println("genpic ok..");
        // 判断如果图形已经存在，就返回 true
        String exists_file = output_filepath + "/" start_dt + "-" + end_dt + "-" + dimtype
+ ".jpeg";
        File picfile = new File(exists_file);
        return picfile.exists();
    }
```

用户在前端检索页面中点击“生成统计图”按钮，就调用了 callRScript 方法，把起止时间作为参数传给 R 脚本，最终生成图形。由于前端是以 Web 页面方式调用的，生成的图形必须放在 Web 容器的 webapp 目录下，才能被查看到。整个交互过程可以采用 AJAX 异步方式，如图 7.5 所示。

3. 使用定时任务生成 R 图

使用 R 语言做数据分析，图表展示是其中的核心功能。爬虫每天从招聘网站抓取的职位信息，如果能够按照特定的维度，每天、每周、每月定期生成 R 图，并自动发布到网上，就会方便用户使用，也增加了数据展示途径，提高了数据利用率。

Hive 职位表中的数据是每天晚上自动导入的，导入后的数据是不会修改的。数据入库后，就可以自动开始执行 R 图的生成功能。统计时间窗可以是当日，也可以是从月初截至当日，由业务人员自己决定。

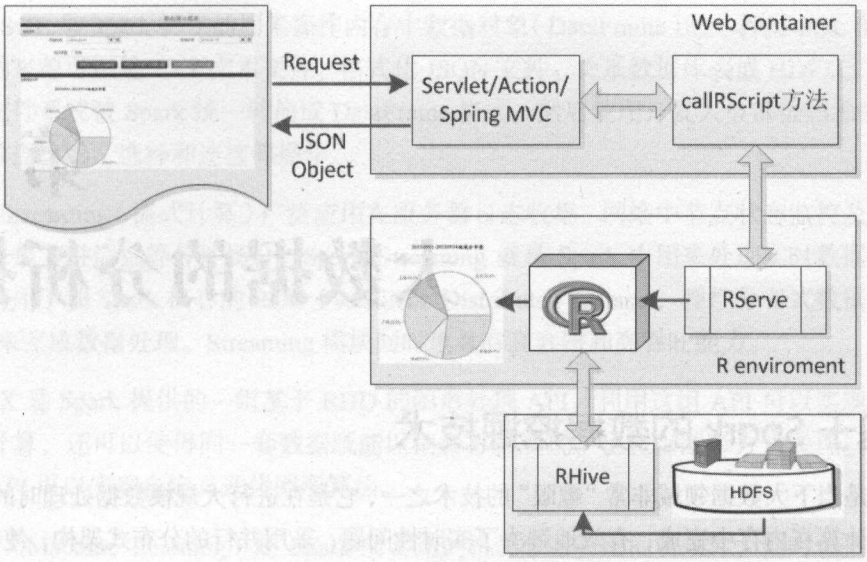


图 7.5　在 Web 端展示 R 图表

上面提到的 callRScript 方法可以照搬过来直接使用，只是起止时间参数会有所调整。生成的图形同样放在 webapp 下的某个目录中，用户可以直接查看而不需要再指定检索条件。自动生成 R 图的功能模块也可以放在 crontab 中进行调度。

在飞谷实践项目中，按照月份统计生成了 4 个维度的 R 图，自动插入到飞谷论坛的飞谷发布模块下，参考网址为 http://www.feiguyun.com/bbs/forum.php?mod=forumdisplay&fid=61。此功能和本章内容关联不大，故不做更多的介绍。

7.5　本章小结

数据展示属于客户需求差异化比较明显的模块，不同的行业和应用场合及访问终端类型不同，决定了数据展示部分有各种表现手段和实现技术。本书是以大数据系统为技术蓝本的，前端采用 R 做图系统，也是其中的一种实现方式。目前市面上关于 R 的专业书籍也有一定的数量，其免费的特性虽然易于传播，但和收费的商业软件相比，入门门槛高和缺少可靠的售后服务是其“短板”。虽然 R 很优秀，但使用范围仅限于数理统计及商业智能（BI）领域。

第 8 章
大数据的分析挖掘

8.1 基于 Spark 的数据挖掘技术

Spark 是当下大数据领域非常“耀眼”的技术之一，它是在进行大规模数据处理时的高效引擎。Spark 数据计算在内存中完成，有效地解决了实时性问题；采用并行的分布式架构，使得用户可以采用大量廉价的设备来提升性能；可伸缩性和健壮性使得它得以稳定运行在不同的生产系统中。

Spark 于 2009 年诞生于 AMPLab，2013 年成为 Apache 的孵化项目，2014 年年初升级为 Apache 的顶级项目，与此同时，Spark 和 Hadoop 的结合，使得它吸引了越来越多的目光。

2015 年 10 月，Spark 版本更新至 1.5.1，2016 年 7 月推出 2.0 版本。主要具有 4 个特点：一是速度快，它通过使用有向无环图（DAG）执行计划，把 Map-Reduce 的计算过程放在内存中完成，比 Hadoop 使用硬盘数据交换方式至少快 10 倍以上；二是上手快，在 Spark 中编写代码，可以选择 Java、Scala、Python、R 语言其中任意一种；三是模块丰富，Spark 提供了 Spark SQL 和 DataFrame、SparkStreaming（流式计算）、SparkGraphX（图计算）以及 Spark 机器学习库（Machine Learning Library）等多种模块，用来满足各种场景下的数据计算需求；四是 Spark 还可以很好地和不同的数据源进行整合，比如 HDFS、HBase、Cassandra、S3 等，充分利用 Spark 计算引擎的特性。

Spark 1.5 包括 4 个功能模块，如图 8.1 所示。

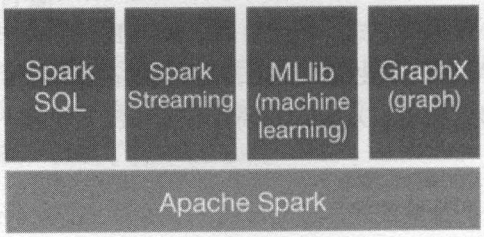


图 8.1　Spark 功能模块

Spark SQL 是 Spark 提供的用来操作内存中数据对象（DataFrame）的一种类 SQL 语言。Spark 处理数据的来源可以是 TXT 文本文件、格式化 JSON 文件、关系数据库表或 HDFS 文件类型，这些异构的文件系统被 Spark 统一转换成 DataFrame 格式，然后使用开发人员都很熟悉的 SQL 语法规则进行数据投影、选择和连接等操作。

Spark Streaming（流式计算）广泛应用在服务器日志收集、网络中节点状态监控及物联网领域从硬件中采集实时信息等各种场合。Spark Streaming 就是 Spark 中用来处理实时数据的模块，它提供了一组用于和 Spark 核心的 RDD（Resilient Distributed Dataset，弹性分布式数据集）进行交互的 API 来完成数据处理。Streaming 模块同时也具有高容错和高吞吐能力。

GraphX 是 Spark 提供的一组基于 RDD 的图形处理 API，利用这组 API 可以实现大规模同步全局的图计算，还可以使得同一套数据既能以集合方式展现，也能以图形方式展现。利用它提供的 Pregel API 可以实现自定义迭代图形算法。

MLlib（Machine Learning）是 Spark 对常用的机器学习算法的实现库，同时包括相关的测试和数据生成器。MLlib 支持各种常见的机器学习问题，如二元分类、回归、聚类以及协同过滤等。由于算法的实现过程使用了 Spark 核心的 RDD，使得在性能方面具有显著的优势。

8.2　Spark 和 Hadoop 的关系

通常来讲，Hadoop 主要包括两大部分，即分布式文件系统 HDFS 和 MapReduce（YARN）计算框架。而 Spark 的核心是一个基于 RDD 数据结构的分布式内存运算模型，它提供了在大数据环境下高效、可靠地完成数据运算的解决方案。

另外，从模块划分上来讲，Hadoop 本身是一个大数据生态系统，包含 Sqoop、Pig、Hive、HBase 等多个组成部分，这些模块分别实现了数据导入导出、数据运算及数据查询等各个方面的功能；而 Spark 所包含的模块，只限定于流式计算、图运算及机器学习等有限的几个方面，并且在这几个应用领域中，也有其他的解决方案对 Spark 构成了有力的挑战。

然而，Spark 受人瞩目的焦点就在于，它对 Hadoop 的数据运算性能有了大幅度的提升（关于这一点，我们在接下来的章节中会做一个性能对比）；并且 Spark 可以和 Hadoop 中的 HDFS、Hive、HBase 等多个模块进行整合，读取其中的数据，从而利用上它的 RDD 内存运算模型。

此外，读者要注意，Spark 并不依赖于 Hadoop，Spark 的核心是一个数据处理模型，它是一个“通用目的”下的集群计算框架。它支持的数据来源是多样的。Hadoop 的 HDFS 只是其中一种数据格式，Spark 是 Hadoop 中计算框架的“升级版本”，二者之间并不是前者要取代后者的关系。

8.3 在 Ubuntu 上安装 Spark 集群

安装 Spark 可以从 Apache 官网下载安装介质，安装方式有两种：一种是直接使用编译好的 Spark 版本；一种是下载 Spark 源码进行编译安装。由于 Spark 主要使用 HDFS 作为数据存储层，所以在安装 Spark 前，要先安装与之对应的 Hadoop 版本。

本书使用的 Spark 版本是 1.4.1，安装文件从（http://www.apache.org/dyn/closer.lua/spark/spark-1.4.1/spark-1.4.1-bin-hadoop2.4.tgz）下载。该 tgz 包中是已经编译好的运行在 Hadoop 2.4 版本上的 Spark。如果用户使用的是其他版本的 Hadoop，也可以下载 Spark 源码后，直接在已经安装好 Hadoop 的环境下手动编译 JAR 包。

Spark 的启动方式有单机环境（本地模式）和集群方式，其中集群方式是真正的生产环境，本节主要描述在集群方式下 Spark 的安装启动步骤。

8.3.1 JDK 和 Hadoop 的安装

这部分内容可以参考“3.2.2 Hadoop 基础环境安装及配置”章节。此外，调试 Spark 程序还可以使用 Python 语言，所以在安装 Spark 前，要保证已经安装好 Python 环境。在“4.2Python 和 Scrapy 框架的安装”章节中，使用的 Python 版本是 2.7.3，这个版本可以和 Spark 1.4.1 兼容使用。

8.3.2 安装 Scala

Scala（scaleable language）是一种语法类似于 Java 语言、以面向对象方式或面向过程编程方式运行的可伸缩语言，它可以和 Java 实现无缝对接。由于 Spark 中使用到了 Scala 语言来操作 RDD，所以要先安装和 Spark 版本对应的 Scala 工具，下载地址为 http://www.scala-lang.org/download/。此处我们使用的版本是 Scala 2.11.6。

安装 Scala 的先决条件是要安装 JDK 1.5 或以上版本，并已经在环境变量中设定好 JAVA_HOME 和将 java\bin 目录添加至系统 PATH 中，然后使用 tar 命令解压缩 scala-2.11.6.tgz 到某个目录下即可。

在/etc/profile 文件中，指定${SCALA_HOME}为刚才解压缩后的文件夹位置，并将 scala\bin 目录添加到系统 PATH 环境变量中，如下所示：

```
export SCALA_HOME=/home/open/bigdata/scala-2.11.6
export PATH=$SCALA_HOME/bin:$PATH
```

配置完成后，在命令行中输入“scala<回车>”，即可进入交互式操作终端，查看当前安装版本号。退出终端使用“:quit”命令。

```
[open@demo ~]$ scala -version
Scala code runner version 2.11.6 -- Copyright 2002-2013, LAMP/EPFL
[open@demo ~]$
[open@demo ~]$ scala
Welcome to Scala version 2.11.6 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0)
Type in expressions to have them evaluated.
Type :help for more information.

scala> :quit
```

另外，需要在集群的各个子节点上都安装相同版本的 Scala。

8.3.3　安装 Spark

安装完 Hadoop 和 Scala 后，就可以安装 Spark 了。通常 Spark 运行在集群环境下，需要依次在 Master 节点和所有的 Slave 节点上都安装 Spark。首先解压缩 Master 机器上下载得到的 spark-1.4.1-bin-hadoop2.4.tgz 文件。

```
tar -xzvf spark-1.4.1-bin-hadoop2.4.tgz
```

进入解压缩后的目录，把 conf 目录下的 spark-env.sh.template 文件复制为 spark-env.sh 文件，并修改该文件的内容。以下是一台 Spark 主机上的配置文件部分内容，安装目录位置仅作为参考。

```
export JAVA_HOME=/home/hadoop/bigdata/java
export SCALA_HOME=/home/hadoop/bigdata/scala
export HADOOP_HOME=/home/hadoop/bigdata/hadoop
export SPARK_MASTER_IP=master
export SPARK_WORKER_INSTANCES=1
export SPARK_MASTER_PORT=7077
export SPARK_MASTER_WEBUI_PORT=8090
export SPARK_WORKER_PORT=7078
export SPARK_WORKER_WEBUI_PORT=8091
export SPARK_WORKER_MEMORY=1000m
export SPARK_DATA_DIR=/home/hadoop/opt/bigdata/data/spark
export SPARK_PUBLIC_DNS=192.168.1.3
export SPARK_DAEMON_JAVA_OPTS=/home/hadoop/bigdata/java
export SCALA_LIBRARY_PATH=/home/hadoop/bigdata/scala/lib
```

其中，SPARK_WORKER_MEMORY 是每个 Worker 节点上 Spark 实例可以使用的系统内存大小，SPARK_MASTER_IP 是 Master 节点的主机名。关于 spark-env.sh 文件中更多配置项的内容释义，可以参考 http://spark.apache.org/docs/latest/configuration.html 文档。

修改完 spark-env.sh 文件后，还要在 conf 目录下创建 slaves 文件，把所有 Worker 节点上的主机名加入 slaves 文件中。示例如下：

```
hadoop@master:~/bigdata/spark/conf$ cat slaves
# A Spark Worker will be started on each of the machines listed below.
slave01
slave02
```

然后可以通过运行 Spark 的测试程序来验证 Master 机器是否安装成功。在 spark\bin 目录下运行 run-example 脚本，如下所示：

```
open@slave01:/spark/bin$ ./run-example SparkPi 10
```

如果运行正常，则可以在日志中找到程序输出的结果信息，打印出 Pi 值。

```
15/10/25 11:10:40 INFO scheduler.TaskSchedulerImpl: Removed Ta
15/10/25 11:10:40 INFO scheduler.DAGScheduler: Job 0 finished:
Pi is roughly 3.144956
```

run-example 脚本中的第一参数 SparkPi，是 spark/lib/spark-examples-1.4.1-hadoop2.4.0.jar 文件中的一个 Java 类名，其作用是利用蒙特卡洛算法计算 Pi 值的示例程序，和 Hadoop 的计算 Pi 值类似。后面的参数“10”是指定首次 Map 的个数，数字越大，计算出的 Pi 值越精确。不同版本的 Spark，调用示例程序所使用的参数有所不同。

此外，在运行示例程序时，可能会抛出 Java 异常，错误信息如下：

```
15/10/25 12:20:25 INFO server.Server: jetty-8.y.z-SNAPSHOT
15/10/25 12:20:25 WARN component.AbstractLifeCycle: FAILED SelectChannelConnect
or@0.0.0.0:4040: java.net.BindException: Address already in use
java.net.BindException: Address already in use
        at sun.nio.ch.Net.bind0(Native Method)
        at sun.nio.ch.Net.bind(Net.java:414)
        at sun.nio.ch.Net.bind(Net.java:406)
```

提示 4040 端口已经被占用。由于每个 Spark 计算任务启动时，都会自动启动 SparkUI 服务，SparkUI 服务是以 Web 方式查看 Spark 任务运行状态的 HTTP 服务。SparkUI 服务默认使用 4040 端口，如果被占用，则会抛出异常，然后尝试使用 4041（4042、4043 等）端口，依此类推。这个异常不影响程序运行。

Master 节点安装完成后，使用“scp”命令把 Master 上 spark 安装目录下的所有内容，复制到每个 Slave 节点相同的目录下，然后就可以测试在集群方式下启动 Spark 服务了。

找到 Master 节点上的 spark\sbin 目录，执行“./start-all.sh”命令，即可启动 Spark 集群，如下所示：

```
hadoop@master:~/bigdata/spark-1.4.1/sbin$ sh start-all.sh
starting org.apache.spark.deploy.master.Master, logging to /home/had
k.deploy.master.Master-1-master.out
slave01: starting org.apache.spark.deploy.worker.Worker, logging to
ache.spark.deploy.worker.Worker-1-slave01.out
slave02: starting org.apache.spark.deploy.worker.Worker, logging to
ache.spark.deploy.worker.Worker-1-slave02.out
```

日志中分别打印出每个 Slave 节点的输出日志文件路径，便于对节点状态进行监控。如果该脚

本没有抛出异常，在 Master 机器上执行“jps”命令，便可以看到 Spark 对应的 Java 进程名字为“8017 Master”，如下所示：

```
hadoop@master:~/bigdata/spark-1.4.1/sbin$ jps
8017 Master
24786 QuorumPeerMain
24390 ResourceManager
24039 NameNode
1848 SparkSubmit
8152 Jps
24729 WebAppProxyServer
24559 JobHistoryServer
```

选择一个 Slave 节点，执行“jps”命令，会看到一个“9196 Worker”进程。

```
hadoop@slave02:~/bigdata/spark-1.4.1$ jps
23680 DataNode
9280 Jps
23794 NodeManager
23946 QuorumPeerMain
9196 Worker
```

Worker 进程是在 Spark 集群模式下，在 Slave 子节点上运行的进程名。

关闭 Spark 集群，只需要在 Master 机器上运行 stop-all.sh 脚本即可。

```
hadoop@master:~/bigdata/spark-1.4.1/sbin$ sh stop-all.sh
slave02: stopping org.apache.spark.deploy.worker.Worker
slave01: stopping org.apache.spark.deploy.worker.Worker
stopping org.apache.spark.deploy.master.Master
```

既然 Spark 可以运行在集群环境下，那么如何在集群上测试 Spark 任务呢？前面我们运行 SparkPi 测试程序，其实是在本地模式（Local）下运行 Spark 任务的。运行本地模式，还可以使用如下命令行语法：

```
./run-example SparkPi 3 spark://master:7077
```

末尾的“spark://master:7077”参数，代表 Spark 集群的 Master 机器 URL。而在集群方式下运行 SparkPi，使用如下命令行：

```
hadoop@master:~/bigdata/spark-1.4.1/bin$ ./spark-submit --master spark://master:7077
--class org.apache.spark.examples.SparkPi --executor-memory 300m  ../lib/spark-exampl
es-1.4.1-hadoop2.4.0.jar 3
```

8.4　Spark 的运行方式

Spark 在设计之初，就考虑到易用性特点，可以和其他语言集成使用。Spark 的运行方式分为命令行交互（shell）方式和脚本（script）方式。在 Spark 中可以使用的语言包括 Scala、Java、Python，

在 Spark 1.4 版本中还支持 R 语言。下面分别列出了三种命令行方式运行 Spark 的方法。为了能在任意目录下直接运行这三种命令，需要把${SPARK_HOME}/bin 目录添加到系统 PATH 环境变量中。

1. spark-shell 命令行

```
hadoop@slave02:~$ spark-shell
15/10/26 11:30:16 WARN NativeCodeLoader: Unable to load native-hadoop li
tform... using builtin-java classes where applicable
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 1.4.1
      /_/

Using Scala version 2.10.4 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.
Type in expressions to have them evaluated.
Type :help for more information.
```

```
SQL context available as sqlContext.

scala> System.out.println("hello spark");
hello spark

scala> println("hello,spark")
hello,spark

scala> exit
```

2. pyspark 命令行

```
open@slave02:~$ pyspark
Python 2.7.3 (default, Dec 18 2014, 19:10:20)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
15/10/26 11:32:04 WARN NativeCodeLoader: Unable to load native-hadoop lib
tform... using builtin-java classes where applicable
```

```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 1.4.1
      /_/

Using Python version 2.7.3 (default, Dec 18 2014 19:10:20)
SparkContext available as sc, HiveContext available as sqlContext.
>>> print('hello spark')
hello spark
>>> quit()
```

3. sparkR 命令行

```
open@slave01:~$ sparkR

R version 3.1.3 (2015-03-09) -- "Smooth Sidewalk"
Copyright (C) 2015 The R Foundation for Statistical Computing
Platform: x86_64-unknown-linux-gnu (64-bit)
```

```
 Welcome to SparkR!
 Spark context is available as sc, SQL context is available as sqlContext
> print("hello,SparkR")
[1] "hello,SparkR"
> q()
Save workspace image? [y/n/c]: n
```

以命令行交互方式操作 Spark，又叫作 Spark shell 方式，shell 方式在启动时，还可以在后面加参数列表，比如“./bin/spark-shell --master local[3]”。关于参数的详细说明，可以参见官方文档（http://spark.apache.org/docs/1.4.1/programming-guide.html#using-the-shell）。

除了使用命令行交互方式外，我们也可以把命令行合并在一个脚本文件中，直接交给 Spark 去执行，就好比把一连串的 Linux 命令行合并在一个 shell 脚本中去执行一样。例如下面两种命令行：

```
$ ./bin/spark-shell --master local[4] --jars code.jar

$ ./bin/pyspark --master local[4] --py-files code.py
```

就是把业务逻辑封装在了 code.jar 和 code.py 中，以文件的方式提交给 Spark 去执行的。

此外，在 8.3.3 节的最后，在 Spark 集群上做 SparkPi 测试时，使用的命令行是“bin$ ./spark-submit”格式，后面加了很多参数。这个脚本是向 Spark 集群提交任务的标准模式，上述的“spark-shell”、“pyspark”和“sparkR”命令执行时，最终都会转换成格式统一的 spark-submit 命令提交。spark-submit 提供了参数列表，用来配置任务执行时的各种环境变量。参数列表如下：

```
./bin/spark-submit \
  --class <main-class>
  --master <master-url> \
  --deploy-mode <deploy-mode> \
  --conf <key>=<value> \
  ... # other options
  <application-jar> \
  [application-arguments]
```

各参数取值的详细说明，请参考官网文档（http://spark.apache.org/docs/1.4.1/submitting-applications.html）。

最后，向 Spark 集群提交任务，还可以集成在应用程序代码中实现。比如在 Java 代码中通过引入 Spark JAR 包，调用 Spark API 方法，在集群中执行特定的任务。这种方式的灵活性最强，接下来的章节中会有详细介绍。

8.5 使用 Spark 替代 Hadoop Yarn 引擎

在 5.1 节介绍 Hive 功能时，大家已经了解到，Hive 是用来以 SQL 方式操作 HDFS 数据的一种工具，在 HQL 进行运算的背后，实际上运行的是 MapReduce 任务。而运行 MapReduce 任务，使用的是 Yarn 引擎。由于在 Yarn 计算模式中，无法避免把运算的中间结果写入磁盘，从而会降低效率。Spark 的基于内存的 RDD 运算模型，很好地解决了这个问题。因此，把这两者结合起来，扬长避短，从 Spark 推出之初，就在努力实现这个目标。

在 Spark 1.0 之前，整合 Apache Hive 和 Spark 使用的是 Shark 项目，即 SQL-on-Spark。在 Spark 1.0 之后，Shark 项目被 Spark SQL 替代，原因是 Shark 代码严重耦合 Hive 中的类，给优化和维护带来了很多不利因素；而 Spark SQL 则完全放弃了 MapReduce 引擎，采用新的查询优化引擎和 RDD 运算，因而使得同样的数据检索动作，使用 Spark SQL 比使用 Hive 速度要快很多倍。

本节就来介绍如何使用 spark-sql，以及在飞谷项目中如何使用 Spark 引擎替代 Hive。

8.5.1 使用 spark-sql 查看 Hive 表

为了能在 spark-sql 交互命令下操作 Hive 表数据，要求本机上同时安装了 Spark 和 Hive 工具，此处用来做演示的 Hive 版本是 0.13.1。首先把 hive-site.xml 文件复制到 spark 的 conf 目录下。

```
hadoop@slave02:~$ cp bigdata/hive/conf/hive-site.xml  bigdata/sp
ark-1.4.1/conf/
```

修改 spark/conf/spark-env.sh 脚本文件，设定本机 Hive 的安装目录。

```
export HIVE_HOME=/home/hadoop/bigdata/hive
```

完成以上简单配置后，就可以在本机上启动 spark-sql 了。由于此处 Hive 的 metaDB 元数据信息是保存在 MySQL 数据库中的，而 spark-sql 启动时会去自动连接 metaDB，所以在命令行参数中，要告诉 spark-sql 脚本 mysql-jdbc 驱动 JAR 包的位置在哪里，使用“--driver-class-path”参数来指定，如下所示：

```
hadoop@slave02:~$ spark-sql --driver-class-path /home/hadoop/big
data/hive/lib/mysql-connector-java-5.1.33-bin.jar
```

在进入 spark-sql 窗口时，最后日志信息会显示出它加载的 Hive 的版本是 0.13.1。

```
SET spark.sql.hive.version=0.13.1
SET spark.sql.hive.version=0.13.1
spark-sql>
```

进入 spark-sql 交互界面后，就可以直接输入 HQL 语句了。下面是一些示例，最后使用“exit”命令退出。

```
spark-sql>
         > show databases;
OK
chiffon
default
demo
feigu3
test
Time taken: 2.984 seconds, Fetched 5 row(s)
spark-sql> use feigu3;
OK
Time taken: 0.131 seconds
```

```
spark-sql> show tables;
daily_dim_sum    false
dim_edu false
dim_job_techword         false
dim_joblocation false
dim_joblocation_test     false
dim_news_techword        false
dim_resume_techword      false
dim_salary       false
dim_workyear     false
dm_job  false
dm_job_test      false
resume_feature   false
rpt_job false
rpt_job_test     false
s_job   false
stg_job false
stg_news         false
stg_resume       false
test     false
Time taken: 0.132 seconds, Fetched 19 row(s)
```

此外，在启动 spark-sql 时，如果不指定“--driver-class-path”参数，也可以在 spark/conf/spark-env.sh 中把 mysql-jdbc 驱动 JAR 包加入类路径中，如下所示：

```
export SPARK_CLASSPATH=$SPARK_CLASSPATH:/home/hadoop/bigdata/hive/li
b/mysql-connector-java-5.1.33-bin.jar
```

8.5.2　在 beeline 客户端使用 Spark 引擎

除了使用 spark-sql 客户端以外，Spark 中也封装了 Hive beeline 客户端的连接方式。beeline 客户端的使用方法在 5.9.1 节中做过介绍，使用该客户端首先要启动 HiveThriftServer 服务。

在 Spark 中也集成了使用 Spark 引擎的 HiveThriftServer 服务，即 sbin 目录下的“start-thriftserver.sh”脚本。在启动服务时，同样需要指定在哪个端口上提供该服务，使用“--hiveconf hive.server2.thrift.port”参数。启动命令行如下所示：

```
hadoop@slave02:~/bigdata/spark-1.4.1/sbin$ ./start-thriftserver.sh
--hiveconf hive.server2.thrift.port=11000
starting org.apache.spark.sql.hive.thriftserver.HiveThriftServer2, l
ogging to /home/hadoop/bigdata/spark-1.4.1/sbin/../logs/spark-hadoop
-org.apache.spark.sql.hive.thriftserver.HiveThriftServer2-1-slave02.
out
```

该服务以后台进程方式启动，日志输出在 spark/logs 目录下。后台进程就是一个 SparkSubmit Java 进程，可以使用“jps”命令查看，其中“10778”是进程编号。

```
23680 DataNode
23794 NodeManager
10900 Jps
21125 Worker
10778 SparkSubmit
23946 QuorumPeerMain
```

关闭该服务，使用“spark/sbin/stop-thriftserver.sh”脚本。

```
hadoop@slave02:~/bigdata/spark-1.4.1/sbin$ ./stop-thriftserver.sh
stopping org.apache.spark.sql.hive.thriftserver.HiveThriftServer2
```

如果服务启动正常，就可以使用 Spark beeline 工具连接 Hive 数据仓库了。同样要在!connect 命令后指明要连接的 IP 地址（主机名）和端口号，在连接成功后的日志信息中，会看到使用的驱动引擎是“Spark Project Core(version 1.4.1)”，如下所示：

```
hadoop@slave02:~/bigdata/spark-1.4.1/bin$ ./beeline
Beeline version 1.4.1 by Apache Hive
beeline> !connect jdbc:hive2://slave02:11000/default
scan complete in 2ms
Connecting to jdbc:hive2://slave02:11000/default
Enter username for jdbc:hive2://slave02:11000/default: hive
Enter password for jdbc:hive2://slave02:11000/default: ****
Connected to: Spark SQL (version 1.4.1)
Driver: Spark Project Core (version 1.4.1)
Transaction isolation: TRANSACTION_REPEATABLE_READ
0: jdbc:hive2://slave02:11000/default>
```

在 beeline 客户端，可以使用 HQL 语句查询和操作 HDFS 中的数据，使用方法与在 Hive beeline 中完全一样。读者可以分别使用 Spark beeline 和 Hive beeline 运行同样的 HQL select 语句，比较两者的速度差别。

```
0: jdbc:hive2://slave02:11000/default> show databases;
+----------+
|  result  |
+----------+
| chiffon  |
| default  |
| demo     |
| feigu3   |
| test     |
+----------+
5 rows selected (2.899 seconds)
0: jdbc:hive2://slave02:11000/default> use feigu3;
+---------+
| result  |
+---------+
+---------+
No rows selected (0.049 seconds)
```

8.5.3　在 Java 代码中引用 Spark 的 ThriftServer

既然说 Spark 引擎比传统的 MapReduce 迭代算法优秀，内存运算比写入硬盘效率高很多，那么有没有实际的例子来做比较呢？

本节就以实现“招聘职位和求职简历自动匹配”功能作为例子，比较两种实现方式到底有哪些差别。

在第 4 章中，我们使用爬虫抓取了很多大数据相关的职位信息，然后使用 Hive 数据仓库做 ETL，最后分析出一些统计指标。对于发布招聘信息的公司 HR 来讲，接下来就是从源源不断的简历中初步筛选出合适的面试人选了。技术类岗位的首要筛选条件，就是专业技能要尽可能匹配，其次是工作年限和期望工资待遇。我们想帮助 HR 实现的就是把招聘岗位中的技术关键字，和求职简历中的专业技能做自动匹配，找出符合技术要求的应聘人员。

此处主要是为了描述技术实现过程，求职简历数据同样使用爬虫从网上抓取，具体的实现步骤不再描述。

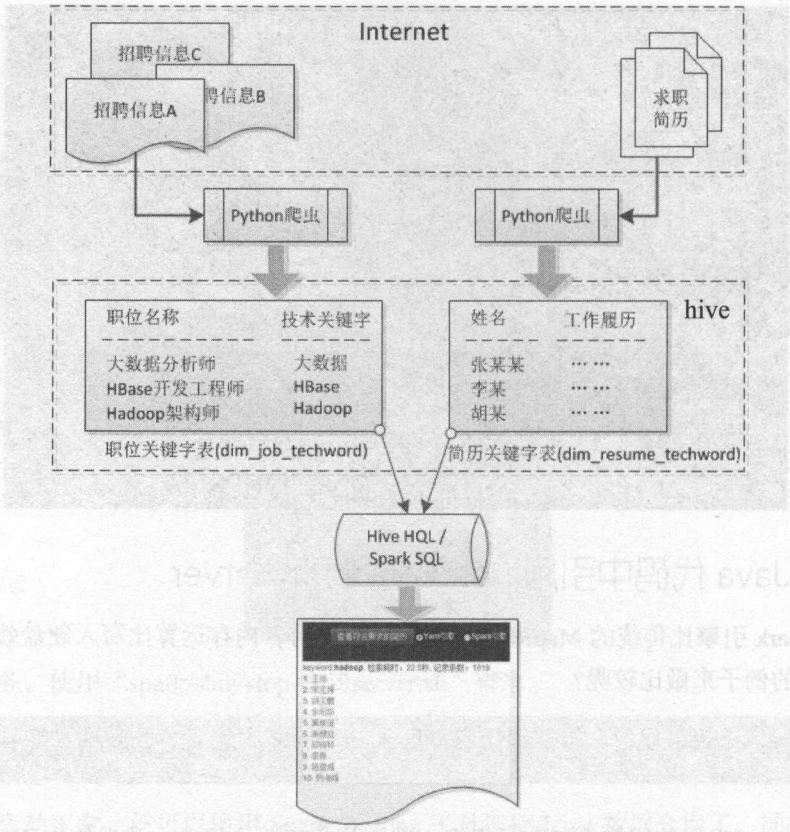


图 8.2　自动匹配简历过程示意图

自动匹配简历涉及三个表，分别是职位关键字表（dim_job_techword）、简历表（stg_resume）和简历关键字表（dim_resume_techword）。三个 Hive 表的逻辑结构如表 8.1、表 8.2、表 8.3 所示。

表 8.1　职位关键字表

字段名	类型	含义	可为空	备注
web_id	String	网站名称	N	变长字符串
job_name	String	职位名称	N	变长字符串
company_name	String	公司名称	N	变长字符串
tech_word	String	技术关键字	Y	变长字符串
pt	String	职位发布日期	N	8 位定长字符串。YYYYMMDD 作为该表的分区，分区名为 pt

表 8.2　简历表

字段名	类型	含义	可为空	备注
web_id	String	来源网站 ID	N	两位定长字符
name	String	姓名	N	变长字符串
update_time	String	最后更新日期	N	8 位字符串。 YYYYMMDD
birth_year	String	生年	N	4 位字符串。YYYY
gender	String	性别	Y	1 位定长字符。M：男，F：女
mobile	String	手机	Y	变长字符串
QQEmail	String	邮箱 QQ	Y	变长字符串
info	String	基本情况	Y	变长字符串
expect_city	String	期望城市	Y	变长字符串
expect_post	String	期望职位	Y	变长字符串
expect_salary	String	期望月薪	Y	变长字符串
work_expr	String	工作经验	N	变长字符串。包括所有的工作经历
work_skill	String	专业技能	N	变长字符串
edu_info	String	学历教育	Y	变长字符串。包括受教育经历
curl_time	String	爬虫抓取时间戳	N	yyyy-mm-dd hh24:mi:ss
curl_link	string	抓取来源网址	N	变长字符串
pt	String	简历抓取时间	N	8 位定长字符串。 YYYYMMDD 作为该表的分区，分区名为 pt

表 8.3　简历关键字表

字段名	类型	含义	可为空	备注
web_id	String	网站标识 ID	N	抓取简历网站标识
name	String	求职者姓名	N	变长字符串
tech_word	String	简历中的技术关键字	Y	
curl_link	String	来源网址	N	
pt	String	抓取日期	N	8 位定长字符串。YYYYMMDD 作为该表的分区，分区名为 pt

职位关键字表的数据来源是 s_job 表，即从 s_job 表的职位描述（job_desc）字段中提取出技术关键字。中文分词和关键字提取，是一个专门的技术领域，需要词库提取算法的支持。此处做简化处理，爬虫抓取职位时，是按照“大数据”、“Hadoop”、“Hive”、“HBase”等 7 个指定的关键字来查找和入库的，分别统计这 7 个关键字在职位描述（job_desc）字符串中出现的次数，把出现频率最高的作为该条招聘信息的技术关键字。以上数据处理过程，在 Hive 中通过 UDF 实现，具体的实现代码不再详细描述。下面是 dim_job_techword 表中部分数据。

```
0: jdbc:hive2://slave01:11000> select * from dim_job_techword
. . . . . .> where pt='20151021' and tech_word is not null
. . . . . .> order by tech_word;
+----------+-------------------------------+-------------------------+-----------+-----------+
| web_type |           job_name            |       company_name      | tech_word |    pt     |
+----------+-------------------------------+-------------------------+-----------+-----------+
| 01       | 大数据开发工程师              | 南京迈特望科技有限公司  | hadoop    | 20151021  |
| 01       | 研究员（大数据分析方向）      | 日立（中国）研究开发有限公司 | hadoop | 20151021  |
| 01       | Hadoop开发工程师-互联网电商   | 数煌网                  | hadoop    | 20151021  |
| 01       | Hadoop高级工程师              | 科技谷（厦门）信息技术有限公司 | hadoop | 20151021  |
| 01       | INTERN - BLUE HIVE Shanghai   | 上海蓝瀚广告有限公司    | hive      | 20151021  |
| 01       | 大数据开发工程师              | 上海透云物联网科技有限公司 | hive      | 20151021  |
| 01       | Hadoop开发工程师              | 海南易建科技股份有限公司（海航集团旗下公司） | 大数据 | 20151021  |
| 01       | 大数据研发工程师              | 宁波搜布信息科技有限公司 | 大数据    | 20151021  |
| 01       | 大数据web开发工程师           | 海南易建科技股份有限公司（海航集团旗下公司） | 大数据 | 20151021  |
| 01       | 大数据开发工程师(职位编号：00257) | 华院数据技术（上海）有限公司 | 大数据 | 20151021  |
| 01       | 大数据产品经理                | 西安习悦信息技术有限公司 | 大数据    | 20151021  |
| 01       | 大数据平台开发工程师          | 北京人大金仓信息技术股份有限公司 | 大数据 | 20151021  |
| 01       | 售前（大数据方向）            | 南京烽火星空通信发展有限公司 | 大数据 | 20151021  |
| 01       | 大数据研发高级经理            | 芒果网有限公司          | 大数据    | 20151021  |
| 01       | IT05 大数据系统开发           | 国泰君安证券股份有限公司 | 大数据    | 20151021  |
| 01       | Hadoop开发工程师              | 北京博睿宏远科技发展有限公司 | 大数据 | 20151021  |
| 01       | 大数据架构师                  | 海南易建科技股份有限公司（海航集团旗下公司） | 大数据 | 20151021  |
| 01       | 大数据项目经理                | 网舟联合科技（北京）有限公司 | 大数据 | 20151021  |
| 01       | 创新工场-诸葛云游-大数据开发工程师 | 创新工场           | 大数据    | 20151021  |
| 01       | 大数据工程师                  | 北京清图杰运软件技术有限公司 | 大数据 | 20151021  |
| 01       | Hadoop中级工程师              | 科技谷（厦门）信息技术有限公司 | 大数据 | 20151021  |
| 01       | 大数据web开发工程师           | 海南易建科技股份有限公司（海航集团旗下公司） | 大数据 | 20151021  |
| 03       | 大数据开发工程师              | 北京盛华合创科技有限公司 | 大数据    | 20151021  |
+----------+-------------------------------+-------------------------+-----------+-----------+
23 rows selected (0.329 seconds)
```

简历表（stg_resume）的数据来源是直接从爬虫生成的文本文件导入到 Hive 中。下面是部分数据。（由于个人信息未作脱敏处理，读者可能看不清楚）

```
hive> select * from stg_resume limit 3;
OK
01      吴湘波<br>      2015-05-28 <br> 53岁<br>        男<br>  NULL    NULL    NULL    北京<br>        机械工程师<br>  12000-20000元/月<br>    10年以上<br>        良好的沟通能力、独立的工作能力、敏捷的思维能力及团队协作精神<br>熟悉产品的研发流程，能依据客户需求数据独立完成产品的总体设计方案，熟悉机械加工工艺及热处理、表面处理流程及相关内容，超强的动手能力，精通Solidworks三维软件，<br>    本科2010.01<br> 2015-06-24 17:27:56     http://jianli.m.58.com/resume/838918237411841/      20150624
01      田丽<br>        2015-06-24 <br> 21岁<br>        NULL    NULL    NULL    NULL    上海浦东、杭州<br>      测试工程师<br>  3000-5000元/月<br>    应届生<br>      本人性格开朗，自信且乐观、脚踏实地、积极上进，具备一定的自学能力，对待工作积极热情，认真负责。忠实诚信，讲原则，说到做到，决不推卸责任；有自制力，做事情始终坚持有始有终，从不半途而废；勤奋好学，有问题不逃避，愿意虚心向他人学习。在生活中能更好的把握自己的时间，充分发挥自己的能量，给大家带来积极乐观的环境。曾在系举办的技能展示大赛获得优秀展示奖，被院校评为优秀大学生。<br>      大专大专大专大专大专国家级二等奖1次2015年5月获得院校级优秀大学生在2013至2014年学度中，表现优异，成绩突出，遵守校纪校规，被院校评为优秀大学生女生生活委员在班级担任女生生活委员，主要负责生活方面的问题，寝室卫生<br>    2015-06-24 17:27:57     http://jianli.m.58.com/resume/845714183787551/   20150624
01      陈望成<br>      2014-08-11 <br> 责任心强<br>    男<br>  NULL    NULL    NULL    广州<br>        软件工程师<br>  12000-20000元/月<br>    3-5年<br>   具有3年从业时间，多年开发经验，在在华为，4399均有做个开发，熟悉java,php,hadoop，c等开发语言，具有深厚的技术功底<br>      本科<br>2007.09-2011.07<br>哈尔滨工程大学  信心与计算科学<br>        2015-06-24 17:27:49     http://jianli.m.58.com/resume/773000046018611/   20150624
Time taken: 1.134 seconds, Fetched: 3 row(s)
```

简历关键字表（dim_resume_techword）的内容来自于简历表，关键字提取方式和招聘职位相同，从简历表的“工作经验”字段中得到。下面是部分数据。

```
01      陈[illegible]成<br>      http://jianli.[illegible].com/resume/773000046018611/   hadoop  20150624
01      莫[illegible]淦<br>      http://jianli.[illegible].com/resume/776288706869811/   hadoop  20150624
01      王[illegible]云<br>      http://jianli.[illegible].com/resume/760597788070471/   hadoop  20150624
01      姜[illegible]<br>        http://jianli.[illegible].com/resume/775914244710441/   hadoop  20150624
01      楼[illegible]燕<br>      http://jianli.[illegible].com/resume/827569231511061/   storm   20150624
```

将职位关键字表和简历关键字表按照关键字做等值连接，即可得到和招聘条件相符合的简历信息。图 8.3 展示了分别使用 Yarn 引擎和 Spark 引擎运行 HQL 后的结果，可以明显地看到后者的速度优势。

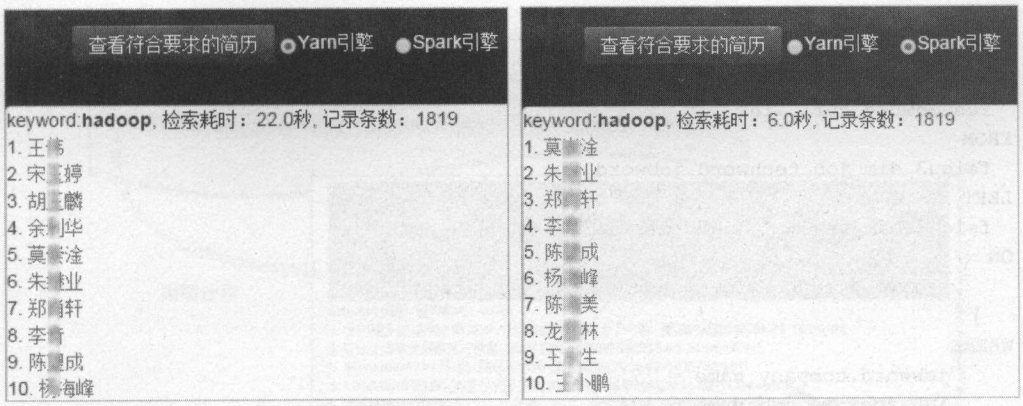


图 8.3　使用 Yarn 引擎和 Spark 引擎运行 HQL 后的结果

那么在 Java 代码中如何使用 Spark 引擎呢?

Java 调用 Spark 有多种方式，其中最简单的就是通过 JDBC。前面已经演示过，beeline 客户端在连接 Hive 数据仓库时，由于事先启动了 Spark 的 HiveThriftServer，那么 beeline 操作数据时就直接使用了 Spark 引擎。而 beeline 本身就是用 Java 语言开发的，使用 JDBC 方式连接 Hive 客户端。此外，在“5.9.3 使用 JDBC 连接 Hive”章节中，也展示过如何使用 Java 代码连接 Hive，我们只需要对代码稍作调整，即可使用 Spark 引擎。

使用 JDBC 连接不同的数据源，只有两处代码需要调整——JDBC 驱动名称和连接字符串格式。下面列出了三种连接 Hive 数据源的不同参数内容。

- HiveServer 方式

JDBC 驱动名称	org.apache.hadoop.hive.jdbc.HiveDriver
JDBC 连接字符串示例	jdbc:hive://slave02:10001/default

- HiveServer2 和 Spark ThriftServer 方式

JDBC 驱动名称	org.apache.hive.jdbc.HiveDriver
JDBC 连接字符串示例	jdbc:hive2://slave01:11000/feigu3

对于后面两种方式，JDBC 的两个参数是完全一样的，即 Spark 是完全兼容 HiveServer2 的。如果要使用 Spark 引擎，只需要先把 spark/sbin/start-thriftserver.sh 脚本运行起来，JDBC 就会自动选择 Spark 引擎来操作 Hive 中的数据了。

最后，无论使用 HiveServer2（即 Yarn 引擎）还是使用 Spark 引擎，JDBC 在操作数据时，都是使用 HQL 语句，这一点是一致的。下面代码演示了如何对职位关键字表和简历关键字表做等值连接。

```
SELECT
  resumeword.tech_word,
  resumeword.name,
  resumeword.curl_link
FROM
  feigu3.dim_job_techword jobword
LEFT OUTER JOIN
  feigu3.dim_resume_techword resumeword
ON (
      jobword.tech_word = resumeword.tech_word
   )
WHERE
     jobword.company_name = '?'
     AND jobword.job_name = '?'
     AND jobword.pt = '?'
```

WHERE 条件中的三个参数值，是由前端页面传入的。由于该 HQL 运行在职位详细页面中，因此公司名称、职位名称和招聘日期都是确定的。检索返回符合条件的求职者名称，点击名称可以跳转到简历详情页面。

8.6 对招聘公司名称做全文检索

Spark 作为一个基于内存的计算引擎，和 Hadoop 之间既有联系又有区别。Spark 既可以和 Hive 配合使用，也可以单独对 HDFS 文件中的数据做运算。而对于大多数数据科学家来讲，直接使用 Scala、Java、Python 语言在 Spark 中进行数据处理，是更加普遍的应用场景。

在 Spark 核心中用来完成迭代算法而构建的数据模型叫作 RDD。RDD（Resilient Distributed Dataset，弹性分布式数据集）是 Spark 主要用来操作的数据对象，充当在多个计算节点上并行运行的分布式数据集合。Spark 核心中的任务调度、内存管理、故障恢复以及和存储介质的交互等功能，都与 RDD 数据结构密不可分。

本节就通过一个示例来说明如何使用 Java 语言对 Spark RDD 进行操作。

首先来描述业务需求。前面我们实现了招聘职位和求职简历按照“技术关键字”匹配的功能。对于查看某条职位信息的求职者而言，除了在详细页面中可以查看到公司描述之外，或许还想关注该公司更多的信息。在 4.8 节中，我们使用 Sqoop 工具，把使用 Scrapy 爬虫抓取的有关大数据方面的新闻信息，从 MySQL 放到了 HDFS 中，接下来可以使用 HQL 导入到 Hive 的表 stg_news 中（具体步骤略）。在招聘职位详情页面，把公司名称作为关键字，在帖子表中做模糊检索，然后把结果异步加载到页面上，就是我们想要实现的功能，如图 8.4 所示。公司名称是“阿里”，检索列出所有和该公司有关的新闻，点击新闻标题，直接跳转到飞谷论坛上的帖子详情页面。

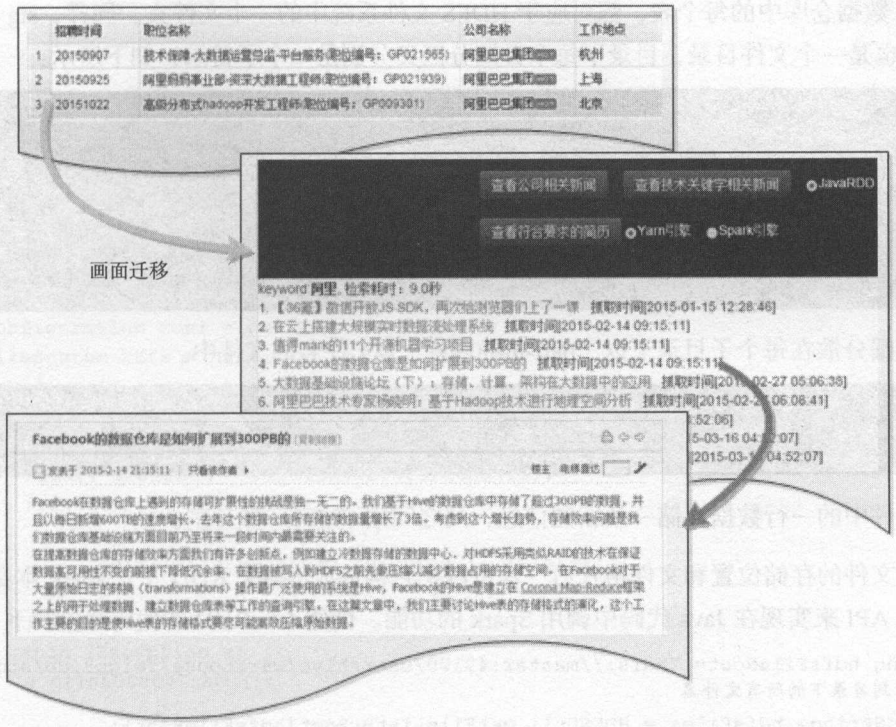


图 8.4　对公司名称做模糊检索

那么，这样一个在传统 RDBMS 中最常用的 LIKE 检索，使用 Spark RDD 又该如何实现呢？

8.6.1　从 HDFS 数据源构造 JavaRDD

先来看如何构造 RDD 的数据源。使用 Sqoop 工具把新闻帖子导入到 HDFS 中后，还是按照之前的思路，进入 Hive 的表 stg_news 中，stg_news 表结构参见“5.3.1 逻辑模型的创建”章节，表中部分数据如下所示：

```
+----------------------+-------------------------------+-------------------------------
-----------------------------------------+
| stg_news.mysql_newsid  |      stg_news.news_title      |
                                          |
+----------------------+-------------------------------+-------------------------------
-----------------------------------------+
| 8935                   | 【CSDN】2014 Docker十大事件回顾       | [p=24, null, left][color=
rgb(51, 51, 51)][font=Helvetica, Tahoma, Arial, sans |
| 8951                   | 【36氪】微信开放JS SDK，再次给浏览器们上了一课  | [b]微信开放JS
SDK，再次给浏览器们上了一课[/b][i]2015-01-14[/i][i]王安[/i][float=left][color=rgb |
| 8952                   | 【大数据】数据将是未来重要的资产              | [b]数据将是未来
重要的资产[/b][color=rgb(119, 119, 119)][backcolor=rgb(249, 249, 249)][fo |
+----------------------+-------------------------------+-------------------------------
```

Hive 数据仓库中的每个表，都对应于 HDFS 文件系统中的一个文件夹。同样，stg_news 表在 HDFS 中也是一个文件目录，目录下每个日期分区又分别是一个子目录，如下所示：

```
open@slave02:~$ hadoop fs -ls /user/hive/warehouse/feigu3.db/stg_news
Found 6 items
drwxr-xr-x   - /user/hive/warehouse/feigu3.db/stg_news/pt=20150101
drwxr-xr-x   - /user/hive/warehouse/feigu3.db/stg_news/pt=20150201
drwxr-xr-x   - /user/hive/warehouse/feigu3.db/stg_news/pt=20150301
drwxr-xr-x   - /user/hive/warehouse/feigu3.db/stg_news/pt=20150401
drwxr-xr-x   - /user/hive/warehouse/feigu3.db/stg_news/pt=20150501
drwxr-xr-x   - /user/hive/warehouse/feigu3.db/stg_news/pt=20150601
```

表数据分散在每个子目录下以“part-m-00000”为文件名的文件中。

```
~$ hadoop fs -ls /user/hive/warehouse/feigu3.db/stg_news/pt=20150501
Found 1 items
/user/hive/warehouse/feigu3.db/stg_news/pt=20150501/part-m-00000
```

该文件中的一行数据存储一条新闻信息，列之间用“\001”做分割。

知道文件的存储位置和文件格式后，就可以把它构造成 Spark RDD 了。Spark 本身提供了丰富的 Java API 来实现在 Java 代码中调用 Spark 的功能。创建 JavaRDD 的代码片段如下：

```
String hdfsFileRoot= "hdfs://master:49100/user/hive/warehouse/feigu3.db/stg_news";
// 得到目录下的所有文件名
List<String> hdfsFiles = HDFSUtil.getFileListByRoot(hdfsFileRoot);

// 构造 Spark 配置信息实例及 Spark SQL 实例
SparkConf sparkConf = new SparkConf().setAppName("newsContentLikeSearch");
sparkConf.setMaster("local");
JavaSparkContext ctx = new JavaSparkContext(sparkConf);
org.apache.spark.sql.SQLContext sqlContext=new org.apache.spark.sql.SQLContext(ctx);

for(int i=0; i<hdfsFiles.size();i++){
   // 把文本文件中的每一行构造成 News 对象，最终形成 Java RDD
   JavaRDD<News> newsSet = ctx.textFile(hdfsFiles.get(i)).map(
   new Function<String, News>() {
      private static final long serialVersionUID = 1L;

      public News call(String line) throws Exception {
         char sep = 01;
         String[] parts = line.split(new Character(sep).toString());
         News news = new News();
         if (parts.length==4){
            news.setNewsId(parts[0]);
            news.setNewsTitle(parts[1]);
            news.setContent(parts[2]);
            news.setCreateTime(parts[3]);
         }
```

```
            return news;
        }
});
```

对于以上代码需要说明几点。

- HDFSUtil.getFileListByRoot 是笔者自定义的递归列出 HDFS 文件系统某个目录下所有文件的方法。使用到了 Java HDFS API 中的 FileSystem 和 FileUtil 类，参考代码如下：

```
public static List<String> getFileListByRoot(String fileRoot){
    List<String> fileLst = new ArrayList<String>();
    Configuration conf = new Configuration();
    FileSystem hdfs = null;
    try {
        hdfs = FileSystem.get(URI.create(fileRoot),conf);
        FileStatus[] fs = hdfs.listStatus(new Path(fileRoot));
        Path[] listPath = FileUtil.stat2Paths(fs);
        for(Path p : listPath) {
          if(p.isUriPathAbsolute()){
             getFileListByPath(p.toString(), fileLst);
          }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    finally {
        try {
            hdfs.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    return fileLst;
}
private static void getFileListByPath(String fileRoot,List<String> files){
  Configuration conf = new Configuration();
  FileSystem hdfs = null;
  try {
    hdfs = FileSystem.get(URI.create(fileRoot),conf);
    FileStatus[] fs = hdfs.listStatus(new Path(fileRoot));
    Path[] listFile = FileUtil.stat2Paths(fs);
    for(Path f : listFile) {
      files.add(f.toString());
    }
  } catch (Exception e) {
  e.printStackTrace();
  }
}
```

- 创建 SparkConf 配置信息实例使用“new SparkConf().setAppName("newsContentLikeSearch")”语法，它为每次 Spark 的运算任务提供初始化参数，与每次创建一个 MapReduce 任务时初始化一个 JobConf 实例类似。setAppName()方法为本次 Spark 运算实例命名，以便在集群中的任务监控页面可以查看到运行状态。setMaster()方法用来设定任务的运行模式，分为两种：“local[4]”表示把任务提交到本机启动 4 个工作线程来运行；“spark://hostName:port”表示把任务提交到集群来运行。除此之外，还可以对本次任务的其他参数进行设定，这些参数的类型和含义，与使用“spark-submit”命令提交任务时，后面所附带的参数释义完全相同，可以参考 8.4 节。
- JavaSparkContext 是在 Java 语言中创建一个 SparkConf 实例。Spark 代码可以使用 Java、Scala、Python 三种语言完成，在这三种语言中，均存在 SparkConf 对象。SQLContext 即 Spark SQL 的上下文配置信息，Spark SQL 是专门用来操作 JavaRDD 的工具。
- “JavaRDD<News> newsSet = ctx.textFile(…).map(…)”这行代码描述了创建一个 JavaRDD 的方式。Spark 提供两种创建 RDD 的方式：从外部数据源导入，或者把一个已经存在的内存集合“并行化（parallelizing）”。此处的 textFile(…)使用的是第一种方式，即从 HDFS 文件中导入数据。
- map(…)中的代码块，使用了 Java 的匿名内部类语法，具体描述如何把数据源中的每一行映射成 Spark RDD 集合中的一个元素。

```
new Function<String, News>() {
    public News call(String line) throws Exception {
        char sep = 01;
        String[] parts = line.split(new Character(sep).toString());
        News news = new News();
        if (parts.length==4){
            // 为每个属性赋值
            news.setNewsId(parts[0]);
            news.setNewsTitle(parts[1]);
            news.setContent(parts[2]);
            news.setCreateTime(parts[3]);
        }
        return news;
    }
}
```

Spark RDD 中的每一个元素都是一个对象。构造该对象，通过实现 org.apache.spark.api.java.function 包中的 Function 接口来完成。基于输入、输出数据类型的不同，Spark API 提供了三种最基本的接口实现方式，如表 8.4 所示。

表 8.4　Spark API 提供的接口实现方式

方法名	实现方式	说明
Function<T,R>	R call(T)	一个输入参数对应一个输出结果。用在 map()或 filter()操作中
Function2<T1, T2, R>	R call(T1,T2)	两个输入参数对应一个输出结果。用在 aggregate()或 fold()操作中
FlatMapFunction<T,R>	Iterable<R> call(T)	一个输入参数对应零或多个输出结果。用在 flatMap()操作中

此处 Map 操作使用的是第一种接口实现方式，即 HDFS 文件中的一行，转换成 RDD 中的一个元素。Function 泛型中 R 的类型定义的是 News 对象。News 对象是一个简单的 Java Bean，其属性定义和 stg_news 表中的字段内容和类型完全一致。代码如下：

```
public class News implements Serializable {

    private static final long serialVersionUID = 1L;

    private String newsId;
    private String newsTitle;
    private String content;
    private String createTime;

    // 每个属性对应一组 set()/get()方法
    ……
}
```

由于已经知道了 HDFS 中数据行的存储格式，在 Function 接口中重写 call()方法时，就对输入参数 String 进行分割，依次放入 News 实例中，最后返回。该实例就映射成 RDD 中的一个元素。

8.6.2　使用 Spark SQL 操作 RDD

以下代码片段，实现了从 RDD 中检索符合条件的公司信息的功能。

```
// 构造 Java RDD
JavaRDD<News> newsSet = ctx.textFile(hdfsFiles.get(i)).map(… …);
newsSet.cache();
// 从 RDD 转换成 DataFrame
DataFrame schemaNews = sqlContext.createDataFrame(newsSet, News.class);
schemaNews.registerTempTable("newsSet");
// 使用 SQL 方法操作 RDD 中的数据
DataFrame relatedNews = sqlContext.sql("SELECT newsId,newsTitle,createTime FROM newsSet
WHERE content LIKE '%" +companyName+"%' ");
```

该段代码中出现了几个 Spark API 中的对象，说明如下：

- JavaRDD<News>使用泛型语法，定义了 RDD 中存储的数据都是 News 对象的实例。“newsSet.cache()”是告诉 Spark 把这个 RDD 缓存到内存中，以备下次直接使用。在 Spark 内存管理功能中，使用 cache()或 persist()方法将可能会反复使用的 RDD 持久化到计算节

点的内存中，是一种通常的做法。在启动 Spark 集群时，可以设定分配给它的总内存数量。在使用时，其 60%用来缓存 RDD，20%用来作为交换内存，剩余部分留给应用程序使用。整个集群可使用的内存数量，在底层还依赖于 JVM 从操作系统中申请到的堆内存大小。

- 为了方便操作 RDD 中的数据，Spark 提供了 DataFrame 对象（在 1.3 版本之前叫作 SchemaRDD）。它是一种以行的方式存储对象，并且包含对象中属性类型描述信息（元数据）的 RDD。类似于传统 RDBMS 中的表，通过 SQL 语句能很方便地操作表中的数据。“registerTempTable()”方法即在集群中把 DataFrame 注册成一个临时表，并命名。
- 在操作 RDD 时，使用了 SQL 语句。SQL 语句的运行需要 SQLContext 对象的支持，这和使用 spark-sql 或 beeline 客户端运行 SQL 需要 SQLContext 是一个道理。不同之处在于，使用 spark-sql，检索的对象是 Hive 数据仓库中的表，而此处检索的对象是内存中的 RDD，故此处使用到的“newSet”表的字段名，是 News 对象中的属性名。

8.6.3 把 RDD 运行结果展现在前端

完成 SQL 的模糊检索后，剩下的就是如何返回结果集了。请看下面的代码：

```
DataFrame relatedNews = sqlContext.sql("… …");
// 把检索结果转换成 JSON 字符串返回
List<String> relatedNewsId = relatedNews.javaRDD().map(new Function<Row, String>() {
    private static final long serialVersionUID = 1L;

    public String call(Row row) {
        String newsId = row.getString(0);
        String title = row.getString(1);
        String ctime = row.getString(2);
        News data = new News();
        data.setCreateTime(ctime);
        data.setNewsId(newsId);
        data.setNewsTitle(title);
        return CommonUtils.produceJson(data);
    }
}).collect();

for(String nid: relatedNewsId){
    resultStr += nid + ",";
}
```

对以上代码分析如下：

- “DataFrame.javaRDD().map(…)”语法是把 SQLContext 运行后得到的结果集——也是一个新的 DataFrame，转换成一个 JavaRDD 的过程。在 map(…).collect()方法内部，同样定

义了一个匿名内部类，该类也实现了 Function 接口，并重写了 call()方法。call()方法的参数是 Row 类型，也就是 RDD 中存储的数据行。

- “row.getString(N)”是取得数据行中第 *N*+1 列数据的值。例如 Spark SQL 中的“SELECT newsId,newsTitle,createTime FROM …”，则 *N* 的取值范围是(0,1,2)。把 RDD 中的数据列依次取出后，放在 News 对象实例中。
- “CommonUtils.produceJson(News)”是笔者自定义的一个方法，用来把 News 实例转换成 JSON 格式字符串，便于在 Web 前端展示。具体代码不再提供。
- “DataFrame.javaRDD().map(…).collect()”语法最后的 collect()方法是对结果集进行收集。Spark 计算框架是分布式的，比如代码中查询出来的 RDD 结果集应该有 10 条记录，这 10 条记录是由集群中的多个节点“分块”运行得到并存储的。而 collect()方法的作用就是把分散的 10 条记录合并到一个节点（即运行 collect()方法的机器）上，合并过程在内存中完成。合并后才可以拿到最终的完整结果集，然后再做后续处理。

通过以上三个步骤，即“取数据→筛选数据→返回结果”，我们完成了一次 Spark RDD 运算的基本过程。读者可能会提出疑问，使用 Spark 内存检索，果真速度就很快吗？这样一个简单的模糊检索，如果在 RDBMS 中很容易就实现了，而使用 Spark 以后，步骤是不是太烦琐了？最后，在 8.5 节中使用 Spark 引擎替代了 Yarn 引擎，在 Hive 中检索也很方便，这和自己手动构造一个 RDD 来检索，哪种方式效率更高、在实际项目中要选择哪种方式呢？

8.7　如何把 Spark 用得更好

提高 Spark 的运行效率，着眼点在两个方面，即“集群”和“内存”。前面在为 SparkConf 配置信息实例赋值时，setMaster()方法中的参数可以指定该任务以何种方式运行——在本机上还是在集群中运行。所有参数的取值和含义，如表 8.5 所示（原文出处请参考 http://spark.apache.org/docs/latest/submitting-applications.html#master-urls）。

表 8.5　setMaster()方法中的参数的取值和含义

取值	含义
local	使用单线程在本机上运行 Spark 任务
local[K]	使用 K 个工作线程在本机上运行 Spark。K 值最好小于等于 CPU 的核数
local[*]	使用和 CPU 核数相同的线程数，在本机上运行 Spark
spark://host:port	连接到 Spark 集群运行任务。host 是集群中的 Master 节点机器名，port 端口号默认是 7077
mesos://host:port	连接到 Mesos 集群运行任务，port 端口号默认是 5055。若 Mesos 集群使用了 ZooKeeper，则格式为 mesos://zk://host:port
yarn-client	以客户端方式连接到 YARN 集群。集群的配置要事先在 HADOOP_CONF_DIR 或 YARN_CONF_DIR 环境变量中设定

续表

取值	含义
yarn-cluster	以集群方式连接到 YARN 集群。集群的配置要事先在 HADOOP_CONF_DIR 或 YARN_CONF_DIR 环境变量中设定

很显然，使用集群方式比单机方式效率更高，有兴趣的读者可以把示例代码中的 setMaster() 方法参数替换成集群方式运行，但前提是要先启动 Spark 集群，然后比较两者的运行耗时。

另外，从 HDFS 中读取数据文件后，构造出来的 RDD 可以使用 persist()或 cache()方法进行持久化。所谓持久化就是数据在内存和硬盘间存储方式的相互转换。由于硬件介质的差异，将数据从硬盘读入内存总是会存在瓶颈，为了便于让 stg_news 表中的数据能够反复使用，Spark 可以让 RDD 一直保持在内存中，通过调用 javaRDD.persist(StorageLevel)方法实现。持久化又分为几个不同的层次，即 StorageLevel 有几种不同的取值。最简单的是全部放入内存，也就是直接使用 javaRDD.cache()方法；也可以同时放入内存和硬盘中，这是考虑内存容量不足的情况；还可以指定持久化集合中的元素是有序的或无序的等。对于那些可能会被反复使用的中间计算结果 RDD，也可以采用持久化技术，目的也是为了提高计算效率。读者可以把上节代码中的 newsSet.cache() 注释掉，来比较两次运行的时间。（注意：持久化要在第二次使用时才能生效）

在分布式环境下，不仅要考虑数据如何运算，同时还要考虑数据如何合并。计算过程是分布式的，但最终结果是要在单个终端展现的。在 8.6.3 节中，计算得到的结果 RDD ——relatedNews，调用了 collect()方法来合并结果集。使用该方法要区分场合，如果运算得到的结果集比较小，则可以使用 collect()方法合并到本地；如果内容很多，超过了本地内存容量，则会出现程序异常。比较稳妥的做法是使用 JavaRDD.take(n)方法，每次只取部分数据，就像实现页面前端的分页显示功能一样。数据运算和合并的过程，对用户来讲是透明的，用户能感觉到的就是一次交互要耗时多少。因此，如何能够尽快地让用户看到结果，也是评判性能优劣的最直观感受。

最后，对于使用 RDD 方式检索，或者使用 Hive+Spark 引擎方式检索，针对单表的查询操作，后者实现起来更方便。对于相同条件下两者的运行速度，笔者也做过简单的比较——stg_news 表中有将近 3000 条数据，对新闻内容做模糊检索，在使用 Spark RDD 单节点模式和 Hive+Spark 引擎方式做一次 SQL 查询时，二者的耗时相差很多，如图 8.5 和图 8.6 所示。

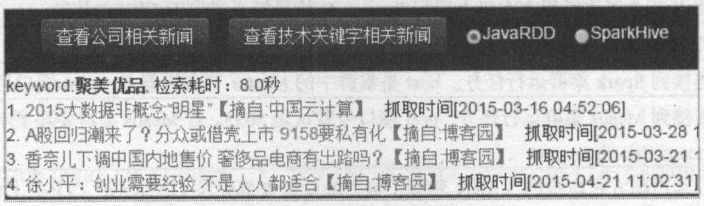


图 8.5 使用 RDD 方式做模糊检索的耗时

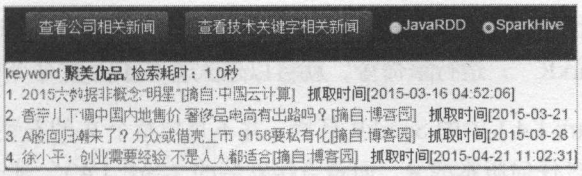


图 8.6　使用 Hive+Spark 引擎方式做模糊检索的耗时

如果加上集群模式和内存持久化，使用 RDD 方式速度也会有所提升，但毕竟代码实现起来没有 Hive 方式方便。那么 RDD 方式的优点到底在哪里？

RDD 是 Spark 计算模型用来运算的通用数据模型，用户可以把各种异构数据源的数据都转换成 RDD 形式。反过来讲，只要想使用 Spark 引擎，就必须按照要求把数据组织成 RDD 方式。另外，Spark 计算框架和 Hive 的着眼点不同。Hive 是 Hadoop 的数据仓库实现，它主要规范了 HDFS 中数据的格式，并且利用 Yarn 引擎来操作其中的数据；就好比在一个砧板上放置了各种食材原料，有荤有素，用一把统一的刀具来切菜。而 Spark 工具就好比一套“瑞士军刀”，有专门切割肉类的尖刀，还有带锯齿的、带锥子的等形状，分别使用在流式处理、图形运算和机器学习等各种场合。而无论在哪种场合下，都是要先把食材分割成统一大小和厚薄的小块，这些小块的集合就是一个 RDD。

8.8　SparkR 组件的使用

Spark 作为一个分布式计算框架，其性能优势是显而易见的；而 R 作为一个数据分析展示的利器，同样拥有众多的“粉丝”。R 的“痛点”在于，R 的 Data Frame 是运行在单节点上的，当数据量特别大时，运算的效率和可靠性会大打折扣。若将两者的优点结合起来，岂不是双赢之选？

迈出第一步的是 R 的开发团队，2014 年 1 月，UC Berkeley's AMPLab 就发布了其 SparkR 项目的开发者预览版，即通过在 R 环境中安装 SparkR 库，让 R 支持 Spark 功能。但安装和使用难度较大，在没有更好的选择之前，这是唯一的解决方法。

而 Apache Spark 则后来者居上，在 2015 年夏天，随着 Spark 1.4 版本的发布，在该版本中包含了 R 前端做图的功能，后端则使用 Spark RDD 来作为强大的数据模型支撑。也就是说，在 Spark 中可以间接地利用 R 中的各种数据挖掘算法，而把性能和扩展性等问题交由 Spark 来处理。所以，Spark1.4 版本一经推出，就受到业界的重点关注，本书中采用 Spark 1.4 也是为了能够使用其提供的 SparkR 功能。

8.8.1　SparkR 的安装及启动

在 8.3.3 节中，我们安装了 Spark1.4.1，该版本的 Spark 已经自带了 SparkR 模块。在 Spark 的安装目录下，可以看到子目录“<SPAKR_INSTALL_PATH>\R\lib”，里面的内容就是在 Spark 上

运行 R 所依赖的包和配置文件。此外，在“<SPAKR_INSTALL_PATH>\bin”目录下，也可以看到可执行脚本文件“sparkR”，运行该命令，就可以进入 R 的交互式命令行环境。

需要注意的是，Spark 1.4.1 中的 SparkR 环境，并不是把整个 R 的安装包都放在了里面，如果读者机器之前没有安装过任何版本的 R，而是直接安装的 Spark 1.4.1，那么运行其中的 sparkR 命令是会出错的，因为 SparkR 只是两者之间的桥梁，并不是替代的关系。

和 Spark 1.4.1 版本相对应的 R 的版本是 3.1.3，更低版本的 R 是无法和 Spark 兼容使用的。另外，Spark 是运行在集群环境中的，集群上每个节点也都要安装 R 环境。

在运行 sparkR 命令前，还要在系统的环境变量中配置两个地方：在 PATH 中要有 ${R_HOME}\bin 目录；同时要增加环境变量 R_HOME，指向安装 R 的本机路径。这两个变量缺一不可，否则运行 sparkR 命令会提示如下错误：

```
hadoop@slave01:~/bigdata/spark/bin$ ./sparkR
env: R: No such file or directory
```

此外，如果是在单机环境下运行 sparkR，要保证机器名（localhost）能够正确映射到本机的 IP 地址，即 127.0.0.1。用户可以修改/etc/hosts 文件，确保“locahost 127.0.0.1”这一行出现在该文件中。如果没有正确映射 localhost，在运行 sparkR 时，会出现“In socketConnection(port = monitorPort) : localhost:xxxxx cannot be opened”此类的错误信息。

另外，sparkR 配置好集群模式后，以单机模式启动，会对配置项中的 hostname 和 IP 地址比较敏感。在单机模式下运行时，可能会出现“Could not connect to akka.tcp://sparkMaster@127.0.0.1:7077: akka.remote.InvalidAssociation: Invalid address: akka.tcp://sparkMaster@127.0.0.1:7077”这样的提示错误。此处，可编辑${SPARK_HOME}/conf/spark-env.xml 文件，注释掉“export MASTER=spark://127.0.0.1:7070”这一行。

如果一切配置正常，在“<SPAKR_INSTALL_PATH>/bin”目录下运行 sparkR 命令，会看到交互界面的启动画面，如下图所示：

```
16/05/12 22:09:44 INFO netty.NettyBlockTransferService: Server created on 43224
16/05/12 22:09:44 INFO storage.BlockManagerMaster: Trying to register BlockManager
16/05/12 22:09:44 INFO storage.BlockManagerMasterEndpoint: Registering block manager
 RAM, BlockManagerId(driver, localhost, 43224)
16/05/12 22:09:44 INFO storage.BlockManagerMaster: Registered BlockManager

 Welcome to SparkR!
 Spark context is available as sc, SQL context is available as sqlContext
>
>
```

首先启动的是 R 的控制台界面，然后自动加载了 sparkR 插件。在该示例中，Spark 已经帮用

户创建好了 sparkR 的上下文环境变量“sc”，用户在退出该界面前，可以手动关闭环境变量，输入命令行“sparkR.stop()”，退出 R 的控制台，输入“q()”命令。

8.8.2　运行自带的 Sample 例子

如果我们打开 sparkR 脚本查看一下，就会发现它其实执行的就是“${SPARK_HOME}/bin/spark-submit”命令，只是后面跟的参数是 R 环境的，这和之前我们演示的计算 PI 值的步骤是殊途同归的。换言之，在 Spark 中运行 R 命令，有两种方式：一种是交互式命令行界面；另一种就是指定 R 脚本的运行方式，比如“spark-submit　R-script-file.R”。

在安装完成 Spark 1.4 后，在${SPARK_HOME}/examples/src/main/r 目录下，有测试 R 脚本文件 dataframe.R，用来测试 sparkR 环境。脚本功能是从 HDFS 中读取 JSON 文件，构造出 sparkRDD，并利用 sparkSQL 来筛选其中符合条件的记录。

我们可以对该示例文件做出一些调整。首先把用来测试的数据文件，即${SPARK_HOME}/examples/src/main/resources/people.json，复制到 HDFS 目录中，比如：

```
hadoop fs -copyFromLocal people.json /spark_text/
```

然后修改 dataframe.R，注释掉之前的文件引用路径，更换成 HDFS 目录下的文件：

```
# Create a DataFrame from a JSON file
#path <- file.path(Sys.getenv("SPARK_HOME"), "examples/src/main/resources/people.json")
path <- file.path("hdfs://master:49100/spark_text/people.json")
```

在${SPARK_HOME}/bin 目录下运行 sparkR 命令，执行该脚本：

```
./sparkR ../examples/src/main/r/dataframe.R
```

最后可以在控制台看到输出的结果：

```
  name
1 Justin
```

8.8.3　利用 SparkR 生成职位统计饼图

除了运行自带的 Sample 例子之外，我们也可以把 Spark-R-Hive 三者结合起来，实现从 Hive 表中读取数据，用 Spark 引擎做数据处理，最后用 R 来做展示。在 7.4 节讲述 RHive 组件时，我们从每日职位维度统计表中，按照不同的维度值生成一个时间段内的饼图，并展现在 Web 端。RHive 中的数据处理引擎，使用的还是 Hadoop MapReduce/Yarn 组件，故此实时性较差。通过 sparkR 组件，可以实现用 Spark 引擎替代 Hadoop，从而缩短图片生成耗时。

整个实现过程还是采用R脚本方式，R代码和7.4.1节的基本一致，只是开头部分的数据获取方式，换成了sparkR模块。代码如下：

```
library(SparkR)
library(Cairo)
sdate<- '20150501'
edate<- '20150531'
dimtype<- 1

sc <- sparkR.init(appName="SparkR-feigu-jobsum")
sparkHiveContext <- sparkRHive.init(sc)

sql(sparkHiveContext, "USE feigu3")
a<-sql(sparkHiveContext, paste('select * from daily_dim_sum where daily_dim_sum.pt
between ',sdate, 'and',edate,sep=" "))
a<-collect(a)
b<-aggregate(a$cnt_val,by=list(a$dim_type),FUN=sum)
……
……
```

对于以上代码片段说明如下：

- 使用控制台方式启动sparkR，会自动创建上下文环境对象“sc”，以方便用户使用；如果是采用R脚本方式创建对象，必须首先加载对应的包，即在程序头标明“library(SparkR)”。
- sparkR.init()方法，手动创建了一个环境对象“sc”，并且用该对象可以继续创建sparkRHive对象，以便打通和Hive数据库的连接。
- 使用sparkRHive对象，可以发送HQL语句，操作Hive表中的数据。
- Spark是运行在集群环境中的，一个HQL语句会在每个计算节点上运算出它所分配的数据量，在使用聚合函数之前，要对每个节点上的数据片段做汇总，可以使用collect()方法实现。

以下是完整的R脚本文件代码。

```
library(SparkR)
library(Cairo)
picpath<- '/home/hadoop/'
sdate<- '20150501'
edate<- '20150531'
dimtype<- 1

sc <- sparkR.init(appName="SparkR-feigu-jobsum")
sparkHiveContext <- sparkRHive.init(sc)

sql(sparkHiveContext, "USE feigu3")
```

```
a<-sql(sparkHiveContext, paste('select * from daily_dim_sum where daily_dim_sum.pt
between ',sdate, 'and',edate,sep=" "))
a<-collect(a)
b<-aggregate(a$cnt_val,by=list(a$dim_type),FUN=sum)
b<-b[order(b[,1]),];
xx<-data.frame(x=c(paste('A',c(1:4,9),sep=""),paste('B',c(1:3,9),sep=""),paste('C',
c(1:3,9),sep=""),paste('D',c(1:3,8:9),sep="")),
y=c('北京','上海','广州','深圳','其他','大专','本科','研究生','其他','3 年以下','3 至 5 年','5
年及以上','其他','一至三万','三至五万','五万以上','面议','其他'));

xx$y<-as.character(xx$y);
b$region<-xx$y[xx$x %in% b[,1]];

##output.filepath 图片输出路径, title 图片标题行
colnames(b)[2]<-'amount';       ##设置列名
b$dim1<-substr(as.character(b[,1]),1,1);
b$ratio<-round(with(b,b$amount*4/sum(b$amount)),3)  ##计算百分比

title<-c('地域','学历','工作经验','月薪')
i<-dimtype;
jpeg(paste(picpath,'/', sdate,'-',edate,'-',dimtype,'.jpeg',sep=''),type="cairo")
pie(b$ratio[b$dim1==unique(b$dim1)[i]],labels=paste(b$region[b$dim1==unique(b$dim1)
[i]],":",b$ratio[b$dim1==unique(b$dim1)[i]]*100,"%",sep=""),family='STKaiti') ##画饼图
title(main=paste(sdate,'-',edate,title[i],'分布图',sep=""),family='STKaiti')  ##设置图
片 title
dev.off()          ##关闭画图

sparkR.stop()
```

8.9　本章小结

由于 Spark 内存运算的高效性和可扩展性，使得越来越多的 IT 企业选择 Spark 来替代基于 MapReduce 或 YARN 的计算框架。本章主要介绍了 Spark 的模块构成、安装过程、和 Hadoop 之间的异同，以及如何使用 Spark 来操作 HDFS 中的数据。

完成 Spark 运算所依赖的数据模型叫作“弹性分布式数据集”，本章通过实现对公司名称做全文检索的需求，展示了如何使用 Java 代码来构建、操作 Spark RDD，并对其中涉及的 Java API 语法做了扼要解释。

为了和本书第 7 章的内容前后呼应，还特别介绍了 Spark 1.4 中引入的新功能——SparkR，希望能在解决大数据展示方面，为读者提供一个新的思路。

第9章 自己动手搭建支撑大数据系统的云平台

随着大数据处理与云计算的发展，要想快速处理庞大的数据，除了掌握大数据处理技术，还需要云计算基础平台技术的支撑。本章就通过大数据处理所用的云基础平台来讲解基于 OpenStack 的飞谷云技术。

通过飞谷云几期云基础平台的实例，熟悉 OpenStack 技术及各组件之间协助的特性，旨在帮助读者快速上手 OpenStack，把相关技术顺利地运用到自己的数据中心中。本章涵盖了云平台部署、参数调试等私有云搭建有关内容，包括：基于大数据处理的需求，如何构建可管理、可维护的平台；从组件安装及配置中获取官方文档中找不到或解决不了的问题答案；最后，根据飞谷项目的不断发展，给出飞谷云平台的自动化技术实现的发展方向。

9.1 云平台架构

飞谷云平台基础架构是建立在飞谷大数据处理项目的特点基础上的，分为：一期架构与二期架构，将来还会有三期、四期架构等。为了能够可持续发展及扩展新技术，我们打算在二期基础上不断丰富平台，增添新组件功能、开发新服务及飞谷自身数据平台的集成等内容。下面根据飞谷建设不同时期的需求，对飞谷云基础平台架构进行说明。

9.1.1 一期云基础平台架构

2014年年中，飞谷公益组织打算构建自己的云基础平台。经过对几种云平台（VMware vSphere、OpenStack、Citrix 等）的实地测试，同时考虑到飞谷云的未来发展，我们最终决定采用 OpenStack（现在比较流行的开源云平台之一）来构建自身的公有云平台。根据数据处理任务对平台及服务

器的性能要求，我们采用了 OpenStack I 版本作为构建云平台的基础。

这期项目作为实验型平台项目，对 OpenStack 诸多基本组件进行了安装与测试，包括：Keystone、Glance、Nova、Swift、Cinder（LVM）、Neutron 及 Horizon。一期云基础平台架构如图 9.1 所示。

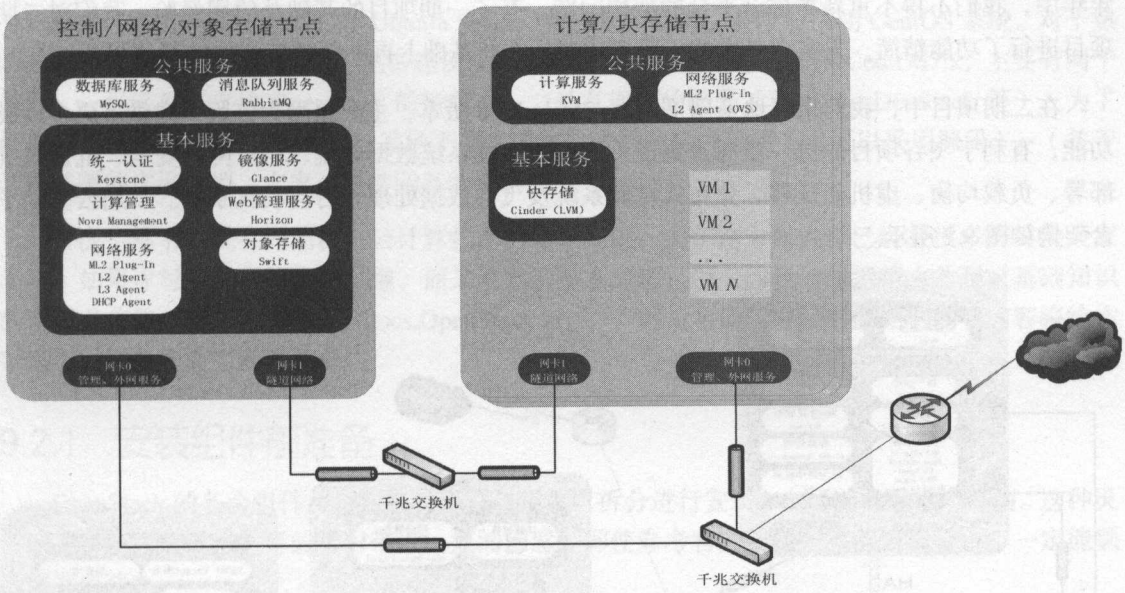


图 9.1　一期云基础平台架构图

在图 9.1 中，为了节约资源，我们采用两台 HP 服务器实现云平台功能。其中，控制功能、网络功能、对象存储功能放在节点 1 上运行，计算功能、块存储功能放在节点 2 上运行。每台服务器有两块网卡：一块作为计算隧道网络使用；一块作为管理网络及外网访问使用。在控制/网络/对象存储节点部分，网卡 1 要配置相关外网访问策略（关于这一点官方文档说明不清晰）；在虚机网络、虚机存储部分都是通过系统内部进行数据交换的。

由于管理网络及外网共用一块网卡，在安全上存在不少问题，因此，我们在 Neutron、物理路由上制定了许多安全策略（二期也是这样的）；虚机的交互数据信息会损耗大量的物理服务器资源（CPU、内存），这个问题在二期中进行了调整。

在一期实验型项目中，安装及部署的 OpenStack 组件并未全部发挥作用；同时，也出现了资源利用率不高的问题。在二期部署文档中提到的内容可以让初学者在构建云平台时避免走弯路，部署出更完善的云环境。

9.1.2 二期云基础平台架构

一期实验型项目的成功部署，让我们实现了数据处理项目从阿里云向飞谷私有云的迁移。通过与阿里云上虚机性能的对比测试，我们发现在相同配置情况下，飞谷云对数据处理的性能更优越些。随着飞谷项目规模的不断扩大，参与的人员越来越多，对服务器的需求也逐渐增加。到 2015 年年中，我们不得不重新考虑云平台的架构问题。有了一期项目的基础及使用经验，我们对二期项目进行了功能精简，先实现基本的云服务功能，在此基础上再不断扩充、增加其他服务。

在二期项目中，我们果断地采用了 OpenStack Kilo 版本，主要是因为该版本提供了更多高级功能，有利于飞谷项目的进一步深入集成。比如：虚机系统数据分流、租户网络安全、批量虚机部署、负载均衡、虚机热迁移、分布式存储系统及飞谷数据处理平台自动化系统。二期云基础平台架构如图 9.2 所示。

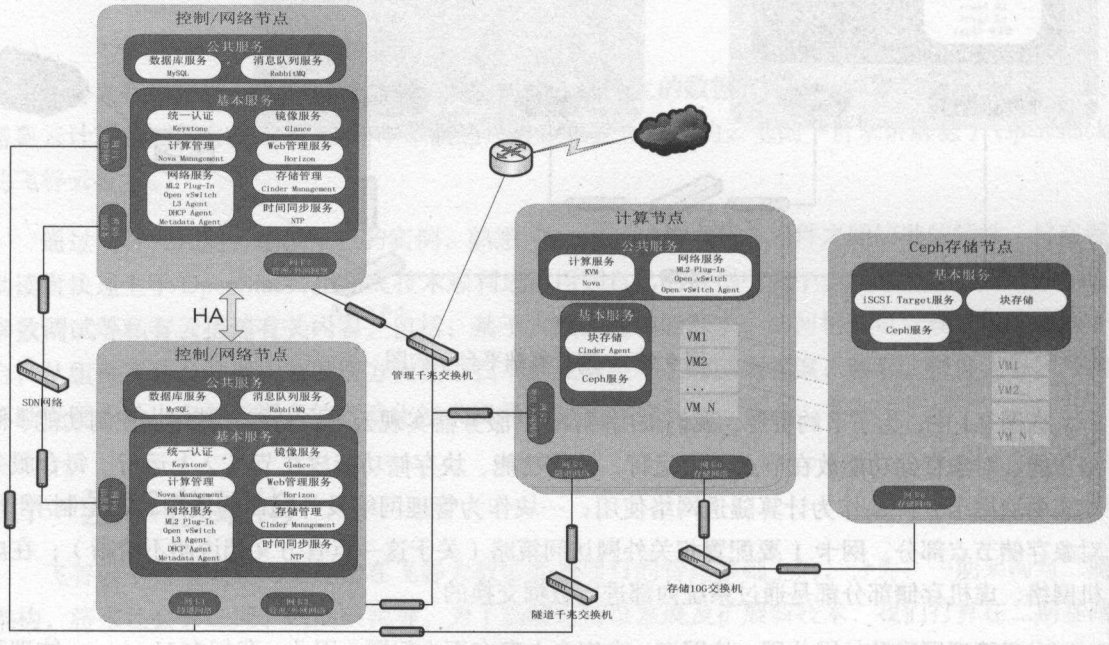


图 9.2 二期云基础平台架构图

在图 9.2 中，控制功能与网络功能集中部署，通过 HA 做控制/网络节点的主备功能，独立进行网络通信；计算节点集群除了提供计算服务外，还用 SSD 磁盘实现虚机性能磁盘的分布式存储；存储节点集群主要为虚机提供数据磁盘功能；计算节点与存储节点集群使用万兆网络进行连接，来提升数据访问速度。因此，总体来说，二期云基础平台架构较为复杂。

此外，租户间虚拟网络也增强了安全策略，用来预防网络安全问题。

9.2　云平台搭建及部署

在 9.1 节中，对飞谷云基本架构做了简单介绍。我们从上面的介绍中可以看出一期、二期项目总体架构的不同，在内部实现及命令上也存在较大的差异。下面根据一期、二期项目中的组件内容进行穿插讲解，以便能在第一时间做对比。

首先，宿主机系统采用的是 Ubuntu Server 14.04 LTS 版本，并没有采用 CentOS 系统。对于熟悉其他 Linux 分支版本的读者，笔者建议采用 CentOS 7.x 作为蓝本。采用 CentOS7.x，主要有两个原因：一是该系统支持 OpenStack 的新版本；二是有现成的部署工具（如：DevStack 等）。为了便于学习研究，我们采用 Ubuntu 系统下的二进制包进行安装及部署（也可以采用源码）。（关于源码部署的详细文档，大家可以参阅其他资料）

其次，本书中没有详细讲解云计算技术的基础知识，关于这方面内容已经有很多资料供大家学习。如果你对云平台技术感兴趣，而又不熟悉基本原理，那么建议你先把精力放到对基础知识的学习上（如：官方文档 http://docs.OpenStack.org/），然后再阅读本书。反之，以下内容将给你带来不一样的技术实现与思考。

9.2.1　安装组件前准备

OpenStack 的各个组件及组件中的服务包都可以拆分进行安装，也可以任意组合使用。这种灵活的构建方法，给初学者的学习带来一定的困惑。即使参考官方文档，摸清原理，也不一定能顺利部署成功。

究竟什么原因呢？首先，OpenStack 涉及的技术知识较多，而且 OpenStack 的各个组件间及组件内部松耦合，使初学者学习起来更加困难；其次，作为初学者，本身技术储备也存在欠缺，需要边学习技术边实践 OpenStack 所需技能。

下面将介绍在安装 OpenStack 组件前，需要提前做好的准备工作。

1. 服务器信息

一期云平台服务器信息，如表 9.1 所示。

表 9.1　一期云平台服务器信息

	CPU	内存	磁盘	OS	节点数量（台）
控制/网络/对象存储节点	Xeon 5 系 v2 (2×2×2)	4×8GB	3×SATA 1TB（RAID 5）	Ubuntu 14.04 LTS	1
计算/块存储节点	Xeon 5 系 v2 (2×2×2)	8×8GB	3×SATA 1TB（RAID 5）	Ubuntu 14.04 LTS	2

说明：在控制/网络/对象存储节点中，SATA RAID 划分成系统盘、对象存储磁盘；在计算/块存储节点中，SATA RAID 划分为系统盘、块存储磁盘。因此，网络拓扑图比较简单，可参考图 9.1。

二期云平台服务器信息，如表 9.2 所示。

表 9.2　二期云平台服务器信息

	CPU	内存	磁盘	OS	节点数量（台）
控制/网络节点	Xeon E5 v2 (2×6×2)	8×16GB	6×SSD 300GB（RAID 1+0）2×SATA 3TB（RAID 1）	Ubuntu 14.04 LTS	2
计算/性能存储节点	Xeon E5 v2 (2×6×2)	32×16GB	2×SATA 1TB（RAID 1） 3×SSD 300GB	Ubuntu 14.04 LTS	5
数据存储节点	Xeon E3 v2 (2×6×2)	8×16GB	2×SSD 300GB（RAID 1） 6×SATA 3TB	Ubuntu 14.04 LTS	3

说明：在控制/网络节点中，SSD RAID 作为本地系统盘，SATA RAID 作为云平台管理平台；在计算/性能存储节点中，SATA RAID 作为本地系统盘，SSD 作为 Ceph 分布式性能盘；在数据存储节点中，SSD 作为本地系统盘（同时加速），SATA 作为 Ceph 分布式数据盘。

两个控制节点采用主备模式，其他节点说明见服务器资源规划图，如图 9.3 所示。

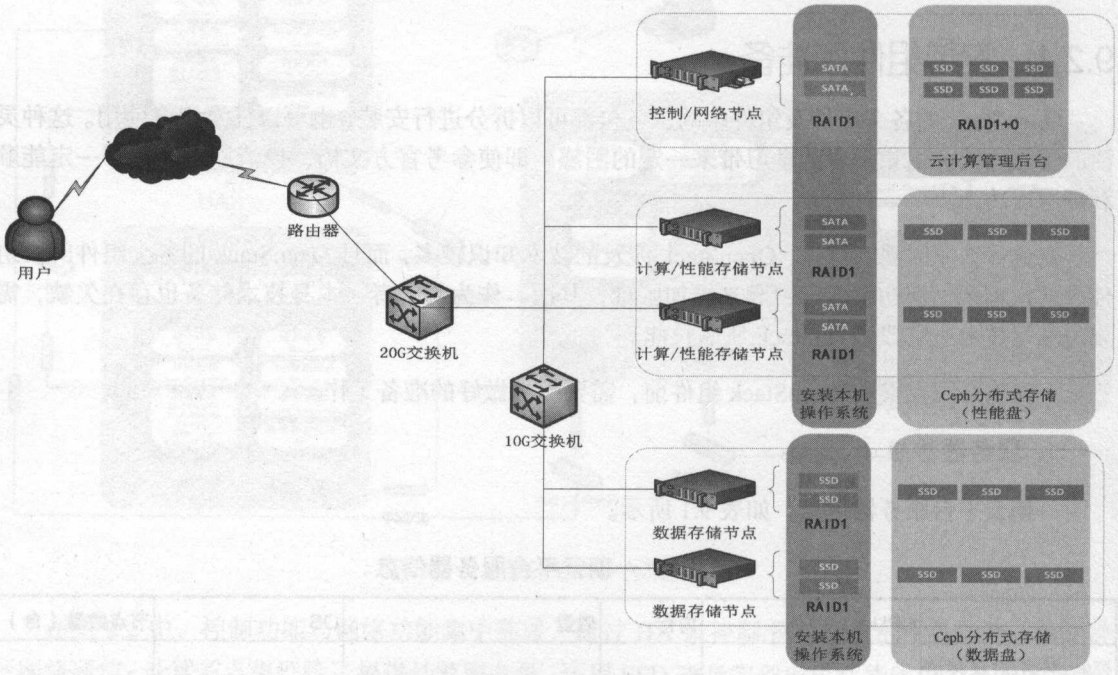


图 9.3　服务器资源规划图

2. 基本环境配置

（1）网络配置（所有节点）

所有节点拥有两块网卡：一块用于控制管理；一块用于虚机的数据流交互。那么，如何配置两块网卡并完成主机名与 IP 地址的解析，尤为重要。

首先，分别在控制节点、计算节点上配置网卡参数。

从一期、二期的云平台架构图可以发现，在控制节点上网卡 0 要实现两个功能：一是管理集群功能；二是作为外网访问虚机功能。在配置控制节点的网卡 0 时，还需要与安装 Neutron 组件配合完成，因此，这一步既重要又复杂。在这里我们配置所有节点的基本网络参数，再次修改控制节点的网卡配置，放到安装 Neutron 组件里。

在所有节点上修改/etc/network/interfaces，配置每个节点的 IP 地址，把*改为自己的所分配的 IP 地址。

```
auto eth0
    iface eth0 inet static
    address 10.3.1.*
    netmask 255.255.255.0
    network 10.3.1.0
    broadcast 10.3.1.255
    gateway 10.3.1.254
    dns-nameservers 114.114.114.114
auto eth1
iface eth1 inet static
    address 10.0.1.*
    netmask 255.255.255.0
    network 10.0.1.0
    broadcast 10.0.1.255
```

网卡 0 作为管理/外网服务接口；网卡 1 作为虚机交互隧道接口。

当服务器数量较少时，为方便节点间通信，通过修改/etc/hosts 来解析主机名与 IP 地址；而当服务器较多时，建议构建内部 DNS 服务器进行解析（注：本解析的 IP 地址段是管理网络中的地址）。在飞谷环境中，采用配置/etc/hosts 来完成主机解析。

```
[root@node01 ~]# echo "10.3.1.55    node05" >> /etc/hosts
[root@node01 ~]# cat /etc/hosts
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6
10.3.1.50       ambaris
10.3.1.51       node01
10.3.1.52       node02
10.3.1.53       node03
10.3.1.54       node04
10.3.1.55    node05
```

上图中，通过多次操作 echo 命令“echo "IP 地址主机名">> /etc/hosts”来添加主机解析信息，

然后用 cat 命令“cat /etc/hosts”来查看配置的解析信息。在一个节点上完成上述配置后，可以使用“scp -r /etc/hosts root@ip:/etc/”命令来完成其他节点上的解析。

最后，通过 ping 主机名命令“ping -c 3 node02”来检查互通情况。

```
[root@node01 ~]# ping -c 3 node02
PING node02 (10.3.1.52) 56(84) bytes of data.
64 bytes from node02 (10.3.1.52): icmp_seq=1 ttl=64 time=0.419 ms
64 bytes from node02 (10.3.1.52): icmp_seq=2 ttl=64 time=0.228 ms
64 bytes from node02 (10.3.1.52): icmp_seq=3 ttl=64 time=0.390 ms
```

（2）配置网络时间同步（所有节点）

在节点上，通过“apt-get install ntp”命令来安装时间同步服务（时间同步是为了保障节点间时间的一致性，便于资源调度）。安装完 NTP 包后，通过修改/etc/ntp.conf 来配置 NTP 服务，命令是：echo"server NTP_SERVER iburst">> /etc/ntp.conf（NTP_SERVER：NTP 服务器 IP 地址，本书中控制节点作为 NTP 服务器）。

然后，重启 NTP 服务，命令是：service ntp restart。

验证 NTP 服务是否同步，命令是：ntpq -c peers。

```
root@node02:~# ntpq  -c peers
     remote           refid      st t when poll reach   delay   offset  jitter
==============================================================================
*202.118.1.130   202.118.1.47     2 u  286 1024  271   96.438  -12.385  28.923
 dns.sjtu.edu.cn 79.213.241.147   3 u   14d 1024    0   29.120  ◦51.075   0.000
 gus.buptnet.edu 202.112.10.60    3 u   55d 1024    0  103.410   28.854   0.000
+news.neu.edu.cn 202.118.1.47     2 u  427 1024  201  158.667  -22.592  11.177
+node01          202.112.29.82    3 u  558 1024  377    0.221   17.749   6.264
```

如果看到列表中出现控制节点（node01）的信息，则说明完成了时间同步服务的配置，否则重新配置该服务。

（3）SSH 信任登录

节点间的信任关系，决定着节点的监控管理，尤其在 OpenStack 的云管理、虚机迁移以及分布式存储等功能上，是非常必要的。

在配置 SSH 信任登录时，为了避免出现对公钥、私钥概念的理解问题，这里采用一种不出错的方式来配置。

首先，在所有节点上配置用户，并使其无密码登录本机系统：

```
# adduser trusty
# echo "trusty ALL = (root) NOPASSWD:ALL" | sudo tee /etc/sudoers.d/trusty
# chmod 0440 /etc/sudoers.d/trusty
```

然后，在控制节点上用 trusty 用户登录系统，并使用以下命令产生密钥：

```
# ssh-keygen
```

接下来，把 SSH 无密码登录运用到其他节点上：

```
# ssh-copy-id trusty@node01
# ssh-copy-id trusty@node02
```

最后，在 trusty 用户目录下添加.ssh/config 文件：

```
# vi ~/.ssh/config
  Host node01
  Hostname node01
  User    trusty
```

依此类推，添加所有节点信息，然后把这个配置文件复制到其他节点的相应位置上，即可完成控制节点到其他节点的无密码登录。

（4）添加 OpenStack 库（所有节点）

Ubuntu14.04 LTS 支持多个 OpenStack 版本。为了部署我们所需的版本，添加获取源有两种方法：

- 搭建本地源服务器并添加资源列表/etc/apt/source.list.d/OpenStack.list。
- 添加官方源。

这里我们采用第二种方法来实现（如果以后增加新节点，应尽量使用本地源服务器，保持版本一致性），在节点上运行以下命令：

```
# apt-get install ubuntu-cloud-keyring
# echo "deb http://ubuntu-cloud.archive.canonical.com/ubuntu"  \
"trusty-updates/kilo main" > /etc/apt/sources.list.d/cloudarchive-kilo.list
```

```
root@node01:~# apt-get install ubuntu-cloud-keyring
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  ubuntu-cloud-keyring
0 upgraded, 1 newly installed, 0 to remove and 85 not upgraded.
```

```
root@node01:~# echo "deb http://ubuntu-cloud.archive.canonical.com/ubuntu"  \
> "trusty-updates/kilo main" > /etc/apt/sources.list.d/cloudarchive-kilo.list
root@node01:~# ls /etc/apt/sources.list.
sources.list.d/    sources.list.save
root@node01:~# ls /etc/apt/sources.list.d/
cloudarchive-kilo.list  webupd8team-java-trusty.list
```

然后，运行下面的命令更新包：

```
# apt-get update && apt-get dist-upgrade
```

（5）安装数据库服务（控制节点）

OpenStack 组件间的数据交互信息最终是要落地将在数据库中的，便于管理各组件间的交互信息。所以，数据库服务是必备的组件，一般将数据库服务安装在控制节点上（如果公司内部有数据库服务，则可以使用）。在该环境中，在控制节点上安装数据库服务，命令如下：

```
# apt-get install mariadb-server python-mysqldb
```

修改数据库配置文件/etc/mysql/conf.d/mysqld_OpenStack.cnf，添加以下内容：

```
[mysqld]
   bind-address = IP(控制节点的 IP 地址)
   default-storage-engine = innodb
   innodb_file_per_table
   collation-server = utf8_general_ci
   init-connect = 'SET NAMES utf8'
   character-set-server = utf8
```

然后，重启数据库服务：

```
# service mysql restart
```

为增强数据库的安全，使用“mysql_secure_installation ”命令进行参数调整。

（6）安装消息队列服务（控制节点）

OpenStack 组件间使用消息队列服务来协同工作或者传递状态信息，消息队列服务在整个云平台虚机信息交互上至关重要。OpenStack 支持的消息队列服务包括 RabbitMQ、Qpid 及 ZeroMQ，可以采用其中任意一种。这里采用的是 RabbitMQ 消息队列服务，主要是因为它在分布式方面支持更好。下面安装消息队列服务及配置该服务：

```
# apt-get install rabbitmq-server
```

添加 OpenStack 用户（用于组件间协同使用，也可以使用其他用户名）并配置访问权限：

```
# rabbitmqctl add_user OpenStack PASSWORD（OpenStack 用户密码）
# rabbitmqctl set_permissions OpenStack ".*" ".*" ".*"
```

至此，基本环境服务安装及配置完毕。

下面将根据一期、二期的安装来对比它们的异同，并进行详细说明与讲解。

9.2.2 Identity（Keystone）组件

在云平台部署中，Identity 组件安装在控制节点上，那么它的重要性是不言而喻的。它的部署

直接成为后续组件能否顺利部署的先决条件。下面我们来完成部署 Identity 组件，并通过其工作原理（示意图如图 9.4 所示）来了解它到底是如何工作的。

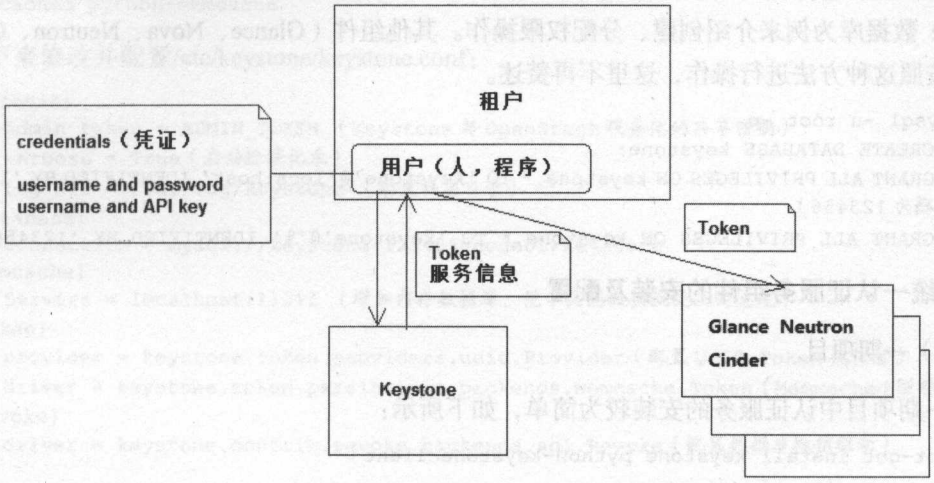


图 9.4　Identity 组建工作原理示意图

OpenStack 的验证服务有两个主要功能：

- 用户管理（租户、用户、权限）。
- 管理服务目录及其 endpoint。

在租户下管理着一些用户（人或程序），每个用户都拥有自己的凭证。用户用凭证去请求 Keystone 服务，获得验证信息（Token 信息）和服务信息（服务目录和 endpoint），然后用户拿着 Token 信息就可以去访问特定的资源了。

在 OpenStack Kilo 版本中，Keystone 命令发生了很多改变。如果对 J 版本之前的 Keystone 命令操作的话，当升级 OpenStack 的相应组件后，需要熟悉新的 Keystone 命令。Keystone 的安装及配置可以通过如下几步来完成：

- 创建统一认证使用的数据库及配置。
- 统一认证服务组件的安装及配置。
- 配置 Web 服务器。
- 创建服务实体及 API 连接点。
- 创建项目、用户及权限。

下面我们根据安装步骤来讲解一期、二期中的各种安装细节。

1. 创建数据库及配置

OpenStack 组件都需要使用数据库，对每个组件创建数据库的方法是相同的，这里以创建 Keystone 数据库为例来介绍创建、分配权限操作。其他组件（Glance、Nova、Neutron、Cinder）都可以按照这种方法进行操作，这里不再赘述。

```
# mysql -u root -p
   CREATE DATABASE keystone;
   GRANT ALL PRIVILEGES ON keystone.* TO 'keystone'@'localhost' IDENTIFIED BY '123456';
（密码为 123456）
   GRANT ALL PRIVILEGES ON keystone.* TO 'keystone'@'%' IDENTIFIED BY '123456';
```

2. 统一认证服务组件的安装及配置

（1）一期项目

在一期项目中认证服务的安装较为简单，如下所示：

```
# apt-get install keystone python-keystoneclient
```

然后修改 Keystone 的配置文件/etc/keystone/keystone.conf：

```
[default]
Admin_token = ADMIN_TOKEN （Keystone 与 OpenStack 服务间的共享密钥）
Log_dir = /var/log/keystone （配置日志目录）
[database]
connection = mysql://keystone:123456@node01/keystone
```

接下来删除默认的本地数据库：

```
# rm -f /var/lib/keystone/keystone.db
```

把配置信息植入 Keystone 服务数据库：

```
# su -s /bin/sh -c "keystone-manage db_sync" keystone
```

重启 Keystone 服务并配置 Keystone 共享密钥的定时日志输出：

```
# service keystone restart
# (crontab -l -u keystone 2>&1 | grep -q token_flush) || \
echo'@hourly/usr/bin/keystone-manage
token_flush>/var/log/keystone/keystone-tokenflush.log2>&1'>>
/var/spool/cron/crontabs/keystone
```

（2）二期项目

首先禁止 Keystone 服务自启动：

```
# echo"manual"> /etc/init/keystone.override
```

然后运行以下命令来安装相关的服务，主要包括 Keystone 服务、Python 的 OpenStack 客户端、

Web 服务器、内存缓冲服务及有关开发语言连接库：

```
# apt-get install keystone python-OpenStack-client apache2 libapache2-mod-wsgi
memcached python-memcache
```

接下来修改并配置/etc/keystone/keystone.conf：

```
[default]
   Admin_token = ADMIN_TOKEN （Keystone 与 OpenStack 服务间的共享密钥）
   verbose = True（启动检修记录）
   Log_dir = /var/log/keystone （配置日志目录）
[database]
   connection = mysql://keystone:123456@node01/keystone
[memcache]
   Servers = localhost:11211 （增加内存数据库，便于提高数据流交互的速度）
[token]
   provider = keystone.token.providers.uuid.Provider（配置 UUID Token 授权者）
   driver = keystone.token.persistence.backends.memcache.Token（Memcached 驱动）
[revoke]
   driver = keystone.contrib.revoke.backends.sql.Revoke（配置数据库撤销驱动）
```

把配置信息植入 Keystone 服务数据库，并删除默认的本地数据库：

```
# su -s /bin/sh -c "keystone-manage db_sync" keystone
# rm -f /var/lib/keystone/keystone.db
```

重启 Keystone 服务：

```
# service keystone restart
```

3. 配置 Web 服务器支持 Keystone 服务

（1）一期项目

在一期项目中，没有配置该功能。

（2）二期项目

在二期项目中，采用 Apache 服务器提供 Keystone 的 Web 认证，具体配置如下。

首先，配置 Apache 服务器主机名：

```
# vi/etc/apache2/apache2.conf
ServerName node01
```

创建/etc/apache2/sites-available/wsgi-keystone.conf，并添加以下内容：

```
Listen 5000
Listen 35357
<VirtualHost *:5000>
   WSGIDaemonProcess keystone-public processes=5 threads=1 user=keystonedisplay-
name=%{GROUP}
```

```
    WSGIProcessGroup keystone-public
    WSGIScriptAlias / /var/www/cgi-bin/keystone/main
    WSGIApplicationGroup %{GLOBAL}
    WSGIPassAuthorization On
    <IfVersion >= 2.4>
       ErrorLogFormat "%{cu}t %M"
    </IfVersion>
    LogLevel info
    ErrorLog /var/log/apache2/keystone-error.log
    CustomLog /var/log/apache2/keystone-access.log combined
</VirtualHost>
<VirtualHost *:35357>
    WSGIDaemonProcess keystone-admin processes=5 threads=1 user=keystone display-
name=%{GROUP}
    WSGIProcessGroup keystone-admin
    WSGIScriptAlias / /var/www/cgi-bin/keystone/admin
    WSGIApplicationGroup %{GLOBAL}
    WSGIPassAuthorization On
    <IfVersion >= 2.4>
    ErrorLogFormat "%{cu}t %M"
    </IfVersion>
    LogLevel info
    ErrorLog /var/log/apache2/keystone-error.log
    CustomLog /var/log/apache2/keystone-access.log combined
</VirtualHost>
```

启动 Keystone 认证虚站点：

```
# ln -s /etc/apache2/sites-available/wsgi-keystone.conf /etc/apache2/sites-enabled
```

创建 WSGI 组件的目录：

```
# mkdir -p /var/www/cgi-bin/keystone
```

从官方复制相应的 WSGI 组件配置到本地目录下，并配置相应文件的权限：

```
#curlhttp://git.OpenStack.org/cgit/OpenStack/keystone/plain/httpd/keystone.py\
?h=stable/kilo | tee /var/www/cgi-bin/keystone/main \
/var/www/cgi-bin/keystone/admin
# chown -R keystone:keystone /var/www/cgi-bin/keystone
# chmod 755 /var/www/cgi-bin/keystone/*
```

重启 Apache 服务：

```
# service apache2 restart
```

4. 创建服务实体及 API 连接点

首先，我们需要配置认证令牌（Token）：

```
$ export OS_TOKEN=ADMIN_TOKEN
$ export OS_URL=http://node01:35357/v2.0
```

（1）一期项目

创建认证服务：

```
$ keystone service-create --name=keystone --type=identity \
--description="OpenStack Identity"
```

创建连接点：

```
$ keystone endpoint-create --service-id=$(keystone service-list \
| awk '/ identity / {print $2}') --publicurl=http://node01:5000/v2.0 \
--internalurl=http://node01:5000/v2.0 --adminurl=http://node01:35357/v2.0
```

（2）二期项目

在创建过程中，命令发生了变化，与早期版本有很大不同：

```
$ OpenStack service create --name keystone --description "OpenStack \
Identity" identity
```

接下来创建 API 连接点：

```
$ OpenStack endpoint create --publicurl http://node01:5000/v2.0 \
--internalurl http://node01:5000/v2.0 --adminurl http://node01:35357/v2.0 \
--region RegionOne Identity
```

5. 创建项目、用户及权限

（1）一期项目

首先创建管理项目：

```
$ keystone tenant-create --name=admin --description="Admin Tenant"
```

然后创建管理用户（使用该命令也可以创建其他用户）：

```
$ keystone user-create --name=admin --pass=123456 --email=admin@126.com
```

再创建一个管理角色，并将该角色分配给管理用户和管理项目：

```
$ keystone role-create --name=admin
$ keystone user-role-add --user=admin --tenant=admin --role=admin
```

同时，在该版本中还需要把管理员加入到默认角色中：

```
$ keystone user-role-add --user=admin --role=_member_ --tenant=admin
```

接下来创建一个服务项目，为后面安装计算服务、镜像服务提供接口：

```
$ keystone tenant-create --name=service --description="Service Tenant"
```

在创建好管理方面的事务后，还需要建立一个普通项目及普通用户：

```
$ keystone user-create --name=test --pass=123456 --email=test@126.com
```

```
$ keystone tenant-create --name=test --description="Test Tenant"
$ keystone user-role-add --user=test --role=_member_ --tenant=test
```

（2）二期项目

首先创建管理项目：

```
$ OpenStack project create --description "Admin Project" admin
```

然后创建管理用户（使用该命令也可以创建其他用户）：

```
$ OpenStack user create --password-prompt admin
```

再创建一个管理员角色，并将该角色分配给管理用户和管理项目：

```
$ OpenStack role create admin
$ OpenStack role add --project admin --user admin admin
```

接下来创建一个服务项目和一个租户项目：

```
$ OpenStack project create --description "Service Project" service
$ OpenStack project create --description "Test Project" test
```

为 Test 项目新建一个用户及角色，并把权限分配给它，采用的命令同上：

```
$ OpenStack user create --password-prompt test
$ OpenStack role create test
$ OpenStack role add --project test-user test test
```

6. 验证 Keystone 服务是否正确

在讲解认证之前，我们首先介绍如何创建客户端脚本，供后面验证使用。

在控制节点上，编辑管理用户（admin）CLI 脚本：

```
$ vi admin.sh
    export OS_PROJECT_DOMAIN_ID=default
    export OS_USER_DOMAIN_ID=default
    export OS_PROJECT_NAME=admin
    export OS_TENANT_NAME=admin
    export OS_USERNAME=admin
    export OS_PASSWORD=123456
    export OS_AUTH_URL=http://node01:35357/v3
```

普通用户（test）与之类似，新建一个 test.sh，复制其上内容，并将所有 admin 替换为 test。

（1）一期项目

配置管理用户获取认证令牌：

```
$ keystone --os-username=admin --os-password=123456 \
--os-auth-url=http://node01:35357/v2.0 token-get
```

另外，管理用户需要获取项目认证权限：

```
$ keystone --os-username=admin --os-password=123456 --os-tenant-name=admin \
 --os-auth-url=http://node01:35357/v2.0 token-get
```

验证是否能获取列表（项目、用户及权限），比如，获取用户列表：

```
$ source admin.sh
$ keystone user-list
```

如果返回列表页面，则说明配置正确；反之，需要检查上面的配置是否存在问题。

（2）二期项目

作为管理用户，需要从认证 API 2.0 中获取认证令牌（Token）：

```
$ OpenStack --os-auth-url http://node01:35357 --os-project-name admin \
--os-username admin --os-auth-type password token issue
```

如果使用 API 3.0，那么还需要获取 3.0 的认证令牌。由于在 API 3.0 中引入了域的概念，因此在授权时命令有所变化：

```
$ OpenStack --os-auth-url http://node01:35357 --os-project-domain-id default \
--os-user-domain-id default --os-project-name admin --os-username admin \
--os-auth-type password token issue
```

作为管理用户，拥有查看项目列表、用户列表及权限列表的权限，而普通用户则没有。可以使用下面的命令实现管理员查看功能：

```
$ OpenStack --os-auth-url http://node01:35357 --os-project-name admin \
--os-username admin --os-auth-type password project list
$ OpenStack --os-auth-url http://node01:35357 --os-project-name admin \
--os-username admin --os-auth-type password user list
$ OpenStack --os-auth-url http://node01:35357 --os-project-name admin \
--os-username admin --os-auth-type password role list
```

但作为普通用户，只能配置获取认证令牌（Token），而没有查看其上列表的权限，除非升级用户等级：

```
$ OpenStack --os-auth-url http://node01:5000 --os-project-domain-id default \
--os-user-domain-id default --os-project-name test --os-username test \
--os-auth-type password token issue
```

验证配置是否正确，如获取用户列表：

```
$ source admin.sh
$ OpenStack userlist
```

如果返回列表信息，则说明配置正确；反之，检查其存在的配置问题。

9.2.3 Image（Glance）组件

在云平台部署过程中，Image 组件一般安装在控制节点上，但也可以单独部署，用来提供多个区域的虚机部署。有了镜像服务，更加方便运维人员批量部署虚机及集成云服务。从这个角度来说，该组件确实简化了很多工作。要安装该组件，首先得从它的工作原理开始了解，如图 9.5 所示。

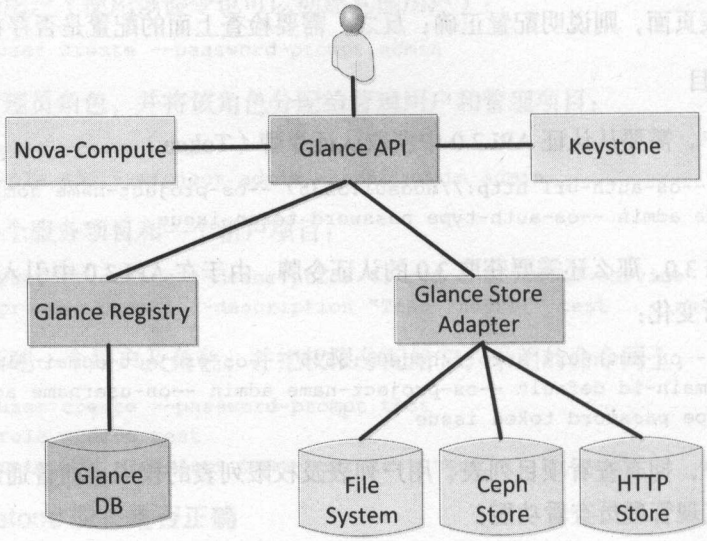


图 9.5 Image 组件工作原理示意图

Image 组件内部主要由两个部件组成。

- glance-api：主要负责接收响应镜像管理命令的 RESTful 请求，分析消息请求信息并分发其所带的命令（如新增、删除、更新等）。
- glance-registry：主要负责接收响应镜像元数据命令的 RESTful 请求。分析消息请求信息并分发其所带的命令（如获取元数据、更新元数据等）。

简单来说，使用 Glance 镜像的原理就是用户通过 glance-api 选择所列出的镜像，到数据库中读取 glance-registry 信息，找到存储信息并返回它的存储位置，然后通过存储驱动调用存储服务器上的镜像来部署虚机获取计算资源的过程。

Image 组件在整个 OpenStack 版本中未发生太大的变化。其安装及配置过程分为以下几步：

- 创建 Glance 数据库及配置（略，见 Keystone 部分）。
- Image 服务组件的安装及配置。

- 创建镜像服务实体项目、用户及权限。
- 验证 Image 服务是否正确。

1. Image 服务组件的安装及配置

安装和配置 Glance 服务，一期、二期项目采用同样的方式，具体如下。

（1）安装 Glance 服务，命令如下：

```
#apt-get install glance python-glanceclient
```

（2）修改 glance-api 配置文件，编辑/etc/glance/glance-api.conf 文件。

① 配置数据库：

```
[database]
connection = mysql://glance:123456@node01/glance
```

② 添加使用认证令牌服务：

```
[default]
auth_strategy = keystone
[keystone_authtoken]
auth_uri = http://node01:5000
auth_url = http://node01:35357
auth_plugin = password
project_domain_id = default
user_domain_id = default
project_name = service
username = glance
password = 123456
[paste_deploy]
flavor = keystone
```

③ 配置镜像存储，这里采用 Ceph 存储镜像文件。其详细参数在 Ceph 安装及配置中讲解。

④ 启动日志功能及通知信息：

```
[default]
verbose = True
notification_driver = noop
```

（3）修改 glance-registry 配置文件，编辑/etc/glance/glance-registry.conf，将 glance-api.conf 文件中添加的[default]、[database]、[paste_deploy]、[keystone_authtoken]的内容原样复制到该文件中。

2. 创建镜像服务实体项目、用户及权限

在创建镜像服务实体项目过程中，与创建 Keystone 服务使用相同的命令。但是由于 OpenStack 版本的问题，命令格式会有所不同。

首先，无论在一期还是二期项目中操作配置，都要先导入管理员访问资源权限：

```
$ source admin.sh
```

接下来，一期项目，分别创建实体、用户及分配权限和连接点：

```
$ keystone user-create --name=glance --pass=123456--email=glance@126.com
$ keystone user-role-add --user=glance --tenant=service --role=admin
$ keystone service-create --name=glance --type=image \
--description="OpenStack Image Service"
$ keystone endpoint-create --service-id=$(keystone service-list \
 | awk '/ image / {print $2}') --publicurl=http://node01:9292 \
--internalurl=http://node01:9292 --adminurl=http://node01:9292
```

二期项目，操作如下：

```
$ OpenStack user create --password-prompt glance
$ OpenStack role add --project service --user glance admin
$ OpenStack service create --name glance -description "OpenStack\
Image service" image
$ OpenStack endpoint create -publicurl http://node01:9292 \
 --internalurl http://node01:9292 -adminurl http://node01:9292 \
 --region RegionOne image
```

在一期、二期项目中，同步数据、删除临时数据库及重启服务命令如下：

```
# su -s /bin/sh -c "glance-manage db_sync" glance
# service glance-registry restart
# service glance-api restart
# rm -f /var/lib/glance/glance.sqlite
```

3. 验证 Glance 服务是否正确

由于 Glance 版本的不同，在上传镜像文件时也存在很大的差异，但总体来说，都是为了把镜像传送到存储中进行保存。首先，还是要导入管理员访问资源权限：

```
$ source admin.sh
```

然后，把镜像导入到分配给 Glance 的存储空间中。

一期项目：

```
$ glance image-create --name "cirros-0.3.3-x86_64" --disk-format qcow2 \
 --container-format bare --is-public True --progress < cirros.img
```

二期项目：

```
$ glance image-create --name "cirros-0.3.3-x86_64" --file \
 /trusty/images/cirros.img --disk-format qcow2 --container-format bare \
 --visibility public --progress
```

最后，检查镜像是否成功上传：

```
$ glance image-list
```

```
trusty@node01:~$ glance image-list
+--------------------------------------+---------------------------------+
| ID                                   | Name                            |
+--------------------------------------+---------------------------------+
| 1c56883b-b242-4d1d-9f4b-843d6cff82c3 | cirros-0.3.3-x86_64             |
+--------------------------------------+---------------------------------+
```

9.2.4　Compute（Nova）组件

在云平台中，Compute 服务可谓是组件中的重中之重，理解 Compute 服务是理解 OpenStack 的基础，它的部署决定了虚机如何获取计算资源。一般来说，Compute 组件控制部分部署在控制节点上，而计算部分必须部署在计算节点上。如果掌握了 Compute 组件的工作原理，那么就更能明白在云平台下虚机是如何获取资源及运行的，在错误排查中也提供了一个参考。Compute 组件工作原理示意图如图 9.6 所示。

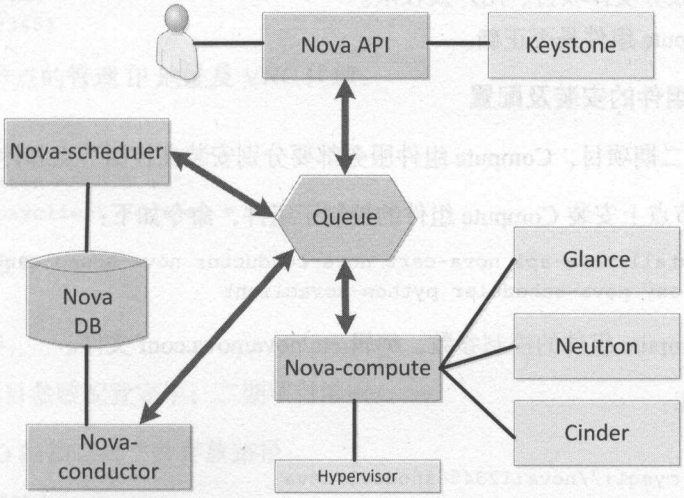


图 9.6　Compute 组件工作原理示意图

Compute 组件内部主要有 4 个部件组成。

- Nova-api：实现了 RESTful API 功能，是外部访问的唯一途径。它接收外部请求并通过消息总线将消息发送给其他的服务组件。
- Nova-conductor：成为 nova-compute 对数据库操作的代理，加强了安全性，提高了性能。
- Nova-scheduler：一个后台进程，主要根据算法从计算资源池选择一个计算节点用于启动虚机。

- Nova-compute：通过消息总线接收虚机生命周期管理指令并实施具体的管理工作，如创建、删除、迁移及改变资源大小等。

从图 9.6 我们知道，Compute 服务中间消息数据较多交换也频繁，因此此处用到了消息队列算法。这个内容我们姑且不深入研究，而是从简单原理讲起——当获取权限的用户通过接口部署虚机时，首先访问数据库获取信息，进入排队环节，等待 scheduler 进程从计算资源中选择合适的计算节点，然后调用 Compute 服务访问数据库数据，获取镜像、网络及存储方面的信息，最后部署到计算节点上。

Compute 组件在 OpenStack 版本中发生了很大变化，子组件不断地从 Compute 组件中分离出来形成新的组件，比如：将 nova-network 子组件合并到 Networking 组件中。接下来我们通过安装及配置来实践剥离后的 Compute 组件，主要分为以下几步：

- 创建 Nova 数据库及配置（略，见 Keystone 部分）。
- Compute 组件的安装及配置。
- 创建计算服务实体项目、用户及权限。
- 验证 Compute 组件是否正确。

1. Compute 组件的安装及配置

无论一期还是二期项目，Compute 组件服务都要分别安装在控制节点和计算节点上。

（1）在控制节点上安装 Compute 组件的控制子组件，命令如下：

```
# apt-get install nova-api nova-cert nova-conductor nova-consoleauth \
 nova-novncproxy nova-scheduler python-novaclient
```

（2）配置 Compute 组件的控制参数，编辑/etc/nova/nova.conf 文件。

① 配置数据库：

```
[database]
connection = mysql://nova:123456@node01/nova
```

② 增加消息队列访问，因为在二期项目中 OpenStack 版本引入了 OSLO，所以在配置上有些不同。

一期项目：

```
[default]
verbose = True
rpc_backend = rabbit
rabbit_host = node01
rabbit_password = 123456
```

二期项目：

```
[default]
rpc_backend = rabbit
[oslo_messaging_rabbit]
rabbit_host = node01
rabbit_userid = OpenStack
rabbit_password = 123456
```

③ 添加使用认证令牌服务：

```
[default]
auth_strategy = keystone
[keystone_authtoken]
auth_uri = http://node01:5000
auth_url = http://node01:35357
auth_plugin = password
project_domain_id = default
user_domain_id = default
project_name = service
username = nova
password = 123456
```

④ 配置计算节点的管理 IP 地址及 VNC 代理：

```
[default]
my_ip = 10.3.1.*
vncserver_listen = 10.3.1.*
vncserver_proxyclient_address = 10.3.1.*
```

⑤ 配置镜像服务：

```
[glance]
Host = node01
```

至此，一期项目参数配置完毕；二期项目继续。

⑥ 配置 OSLO 的锁临时文件存放路径：

```
[oslo_concurrency]
lock_path = /var/lib/nova/tmp
```

（3）在计算节点上安装 Compute 组件的计算子组件。由于一期、二期项目版本不同，组件的名称也发生了改变，因此安装存在不同。

一期项目：

```
# apt-get install nova-compute-kvm
```

二期项目：

```
# apt-get install nova-compute sysfsutils
```

① 在计算节点上，对计算组件的配置如同控制节点配置，但也有不同点。

我们先把控制节点配置中的①、②、③、⑤内容直接复制到/etc/nova/nova.conf 文件中（二期项目版本还要复制⑥内容），然后在配置管理 IP 地址与 VNC 时发生改变，具体添加内容如下：

```
[default]
my_ip = 10.3.1.* （计算节点管理口 IP 地址）
vnc_enabled = True
vncserver_listen = 0.0.0.0
vncserver_proxyclient_address = 10.3.1.*
novncproxy_base_url = http://对外 IP:6080/vnc_auto.html
```

注：对外 IP 地址可以是控制节点 IP 地址（官方配置文档采用这个），或者是映射到公网的 IP 地址或域名。主要是让用户通过外网访问 Web 端虚机终端。

② 在计算节点上指定要启动的虚拟类型，我们先使用如下命令进行测试：

```
$ egrep -c '(vmx|svm)' /proc/cpuinfo
```

```
trusty@node01:~$ egrep -c '(vmx|svm)' /proc/cpuinfo
24
```

如果返回“0”，则说明要采用 QEMU；返回“非 0”，则会配置成 KVM。

然后配置/etc/nova/nova-compute.conf：

```
[libvirt]
virt_type = kvm
```

最后删除临时数据库并重启服务：

```
# rm -f /var/lib/nova/nova.sqlite
# service nova-compute restart
```

2. 创建计算服务实体项目、用户及权限

在控制节点上创建计算服务实体项目、用户及权限的过程与创建 Glance 相关服务一样，也是因为 OpenStack 版本问题，命令格式有所不同。

首先导入管理员访问资源权限：

```
$ source admin.sh
```

接下来，一期项目，分别创建实体、用户及分配权限和连接点：

```
$ keystone user-create --name=nova --pass=123456--email=nova@126.com
$ keystone user-role-add --user=nova --tenant=service --role=admin
$ keystone service-create --name=nova --type=compute\
--description="OpenStack Compute"
$ keystone endpoint-create \
--service-id=$(keystone service-list | awk '/ compute / {print $2}') \
```

```
--publicurl=http://node01:8774/v2/%\(tenant_id\)s \
--internalurl=http://node01:8774/v2/%\(tenant_id\)s \
--adminurl=http://node01:8774/v2/%\(tenant_id\)s
```

二期项目，操作如下：

```
$ OpenStack user create --password-prompt nova
$ OpenStack role add --project service --user nova admin
$ OpenStack service create --name nova-description "OpenStack\
 Compute" compute
$ OpenStack endpoint create \
--publicurl http://node01:8774/v2/%\(tenant_id\)s \
--internalurl http://node01:8774/v2/%\(tenant_id\)s \
--adminurl http://node01:8774/v2/%\(tenant_id\)s \
--region RegionOne compute
```

在一期、二期项目中，同步数据、删除临时数据库及重启服务命令如下：

```
# su -s /bin/sh -c "nova-manage db sync" nova
# service nova-api restart
# service nova-cert restart
# service nova-consoleauth restart
# service nova-scheduler restart
# service nova-conductor restart
# service nova-novncproxy restart
# rm -f /var/lib/nova/nova.sqlite
```

3. 验证 Compute 组件是否正确

在控制节点上导入管理员访问权限，然后通过命令来检查组件是否安装成功。

```
$ source admin.sh
$ nova service-list
```

```
trusty@node01:~$ nova service-list
+----+------------------+--------+----------+---------+-------+----------------------------+-----------------+
| Id | Binary           | Host   | Zone     | Status  | State | Updated_at                 | Disabled Reason |
+----+------------------+--------+----------+---------+-------+----------------------------+-----------------+
| 1  | nova-cert        | node01 | internal | enabled | up    | 2016-03-20T14:35:24.000000 | -               |
| 2  | nova-consoleauth | node01 | internal | enabled | up    | 2016-03-20T14:35:25.000000 | -               |
| 3  | nova-scheduler   | node01 | internal | enabled | up    | 2016-03-20T14:35:22.000000 | -               |
| 4  | nova-conductor   | node01 | internal | enabled | up    | 2016-03-20T14:35:27.000000 | -               |
| 5  | nova-compute     | node01 | nova     | enabled | up    | 2016-03-20T14:35:25.000000 | -               |
| 6  | nova-compute     | node02 | nova     | enabled | up    | 2016-03-20T14:35:26.000000 | -               |
+----+------------------+--------+----------+---------+-------+----------------------------+-----------------+
```

```
$ nova image-list
```

```
trusty@node01:~$ nova image-list
+--------------------------------------+---------------------+--------+--------+
| ID                                   | Name                | Status | Server |
+--------------------------------------+---------------------+--------+--------+
| 1c56883b-b242-4d1d-9f4b-843d6cff82c3 | cirros-0.3.3-x86_64 | ACTIVE |        |
+--------------------------------------+---------------------+--------+--------+
```

9.2.5 Storage（Cinder）组件

在 OpenStack 持久化存储组件中包括了：Cinder（块存储）和 Swift（对象存储），这里我们深入讲解 Cinder（块存储）。

Cinder 是一个资源管理系统，负责存储资源的分配，对不同的后端存储进行封装，向外提供统一的 API。OpenStack 没有开发块设备存储系统，Cinder 只是结合不同后端存储的 Driver 提供块设备存储服务。从这个角度理解就容易多了。那么，该组件如何工作呢？我们需要从它的工作原理说起，如图 9.7 所示。

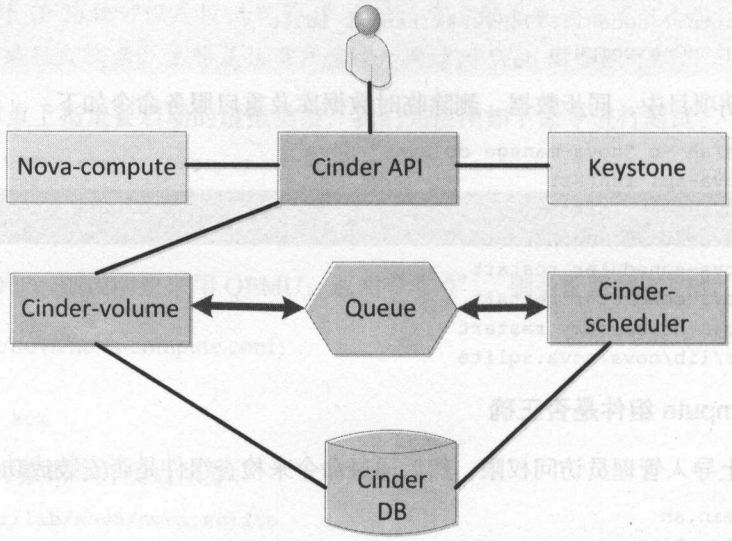


图 9.7　Storage 组件工作原理示意图

Storage 组件内部主要有三个部件组成。

- Cinder-api：用来接收用户请求，控制 Cinder-volume 的路由行动。
- Cinder-scheduler：负责分配存储资源，就是选择一个可以创建卷的存储节点。
- Cinder-volume：负责封装 Driver，不同的 Driver 负责控制不同的后端存储。

简单来说，使用 Storage 组件的原理就是用户通过 Cinder-api 接收用户请求，将请求加入到队列中；然后 Cinder-scheduler 处理队列任务，根据预定策略选择合适的 Cinder-volume 服务节点来执行任务；最后由运行在存储节点上的 Cinder-volume 管理、分配存储空间供虚机使用。

Storage 组件的安装及配置过程，分为如下几步：

① 创建 Cinder 数据库及配置（略）。

② Storage 组件的安装及配置。

③ 创建块存储服务实体项目、用户及权限。

④ 验证 Storage 组件是否正确。

1. Storage 组件的安装及配置

无论在一期还是二期项目的安装及部署中，Storage 服务都要根据提供的功能不同，把相应的子组件安装在控制节点和计算节点上（在现在的项目中计算节点也是存储节点）。

（1）为了便于管理和维护，把 Storage 组件的控制部分安装在控制节点上，命令如下：

```
# apt-get install cinder-api cinder-scheduler python-cinderclient
```

（2）配置相关参数，编辑/etc/cinder/cinder.conf 文件。接下来的内容与 Compute 服务的配置大致相同。

① 配置数据库：

```
[database]
connection = mysql://cinder:123456@node01/cinder
```

② 增加消息队列访问。

一期项目：

```
[default]
verbose = True
rpc_backend = rabbit
rabbit_host = node01
rabbit_port = 5672
rabbit_userid = guest
rabbit_password = 123456
```

二期项目：

```
[default]
rpc_backend = rabbit
[oslo_messaging_rabbit]
rabbit_host = node01
rabbit_userid = OpenStack
rabbit_password = 123456
```

③ 添加使用认证令牌服务。

一期项目：

```
[default]
auth_strategy = keystone
```

```
[keystone_authtoken]
[keystone_authtoken]
auth_uri = http://node01:5000
auth_host = node01
auth_port = 35357
auth_protocol = http
admin_tenant_name = service
admin_user = cinder
admin_password = 123456
```

二期项目：

```
[default]
auth_strategy = keystone
[keystone_authtoken]
auth_uri = http://node01:5000
auth_url = http://node01:35357
auth_plugin = password
project_domain_id = default
user_domain_id = default
project_name = service
username = cinder
password = 123456
```

④ 配置管理 IP 地址：

```
[default]
my_ip = 10.3.1.*
```

接下来的内容只有二期项目需要配置。

⑤ 配置 OSLO 的锁临时文件存放路径：

```
[oslo_concurrency]
lock_path = /var/lib/cinder
```

（3）在计算节点上安装 Cinder-volume，并编辑/etc/cinder/cinder.conf 的相关参数：

```
# apt-get install cinder-volume python-mysqldb
```

在计算节点上，配置存储参数与控制节点有相同的地方，也有不同的地方。首先，将控制节点配置中的①、②、③、④内容复制到/etc/cinder/cinder.conf 文件中（二期项目版本还要增添⑤内容）。然后，还要在配置文件中添加以下内容：

```
[default]
glance_host = node01
```

以及配置访问分布式存储系统（Ceph）的驱动及访问权限，详见 Ceph 章节。

如果你使用的是 LVM 存储，那么配置就会发生改变，详情请参照官方安装文档。

最后，删除临时数据库及重启服务：

```
# rm -f /var/lib/cinder/cinder.sqlite
# service cinder-volume restart
# service tgt restart
```

注：关于 Ceph 配置内容，请参照 Ceph 安装及配置章节。

2. 创建块存储服务实体项目、用户及权限

在控制节点上创建块存储服务实体项目、用户及权限的过程与其他组件相同。

首先，导入管理员访问资源权限：

```
$ source admin.sh
```

接下来，一期项目，分别创建实体、用户及分配权限和连接点：

```
$ keystone user-create --name=cinder --pass=123456--email=cinder@126.com
$ keystone user-role-add --user=cinder --tenant=service --role=admin
$ keystone service-create --name=cinder --type=volume\
--description="OpenStackBlock Storage"
$ keystone service-create --name=cinderv2 --type=volumev2 \
--description="OpenStack Block Storage v2"
$ keystone endpoint-create \
--service-id=$(keystone service-list | awk '/ volume / {print $2}') \
--publicurl=http://node01:8776/v1/%\(tenant_id\)s \
--internalurl=http://node01:8776/v1/%\(tenant_id\)s \
--adminurl=http://node01:8776/v1/%\(tenant_id\)s
$ keystone endpoint-create \
--service-id=$(keystone service-list | awk '/ volumev2 / {print $2}') \
--publicurl=http://node01:8776/v2/%\(tenant_id\)s \
--internalurl=http://node01:8776/v2/%\(tenant_id\)s \
--adminurl=http://node01:8776/v2/%\(tenant_id\)s
```

二期项目，操作如下：

```
$ OpenStack user create --password-prompt cinder
$ OpenStack role add --project service --user cinder admin
$ OpenStack service create --name cinder-description "OpenStack\
 Block Storage" volume
$ OpenStack service create --name cinderv2-description "OpenStack\
 Block Storage" volumev2
$ OpenStack endpoint create \
--publicurl http://node01:8776/v2/%\(tenant_id\)s \
--internalurl http://node01:8776/v2/%\(tenant_id\)s \
--adminurl http://node01:8776/v2/%\(tenant_id\)s \
--region RegionOne volume
$ OpenStack endpoint create \
--publicurl http://node01:8776/v2/%\(tenant_id\)s \
```

```
--internalurl http://node01:8776/v2/%\(tenant_id\)s \
--adminurl http://node01:8776/v2/%\(tenant_id\)s \
--region RegionOne volumev2
```

同步数据、删除临时数据库及重启服务命令如下：

```
# su -s /bin/sh -c "cinder-manage db sync" cinder
# service cinder-scheduler restart
# service cinder-api restart
# rm -f /var/lib/cinder/cinder.sqlite
```

3. 验证 Storage 组件是否正确

在控制节点上导入管理员访问权限，然后检测 Storage 组件是否安装成功。

```
$ source admin.sh
$ cinder create --display-name myVolume 1
```

```
trusty@node01:~$ cinder create --display-name myVolume 1
+---------------------------------------+--------------------------------------+
|                Property               |                Value                 |
+---------------------------------------+--------------------------------------+
|             attachments               |                  []                  |
|          availability_zone            |                 nova                 |
|               bootable                |                false                 |
|          consistencygroup_id          |                 None                 |
|              created_at               |      2016-03-21T14:16:31.000000      |
|             description               |                 None                 |
|              encrypted                |                False                 |
|                  id                   | ed82251a-0146-4647-9456-dce709374da4 |
|              metadata                 |                  {}                  |
|             multiattach               |                False                 |
|                  name                 |               myVolume               |
|       os-vol-host-attr:host           |         node02@rbd#RBD_iSCSI         |
```

```
$ cinder list
```

```
trusty@node01:~$ cinder list
+--------------------------------------+-----------+----------+------+-------------+----------+-------------+
|                  ID                  |   Status  |   Name   | Size | Volume Type | Bootable | Attached to |
+--------------------------------------+-----------+----------+------+-------------+----------+-------------+
| ed82251a-0146-4647-9456-dce709374da4 | available | myVolume |  1   |     None    |  false   |             |
+--------------------------------------+-----------+----------+------+-------------+----------+-------------+
```

9.2.6 Networking（Neutron）组件

俗话说，在 OpenStack 中搞定了网络部分就搞定了 OpenStack。不言而喻，网络组件在 OpenStack 所有组件中的重要性及其复杂性。在安装及配置网络部分，也一度让初学者很头痛、很困惑，笔者也曾遇到同样的问题。因此，我们会用更多的篇幅来讲解 Neutron 网络部分内容（关于 Nova-network，请参考官方部署文档）。

从官方可知 Neutron 是“网络即服务”的组件，主要提供云计算环境下虚拟网络功能。随着

Networking 的发展，在 Horizon 中进行了集成而且功能也在不断增加。下面我们看看它的基本工作原理，如图 9.8 所示。

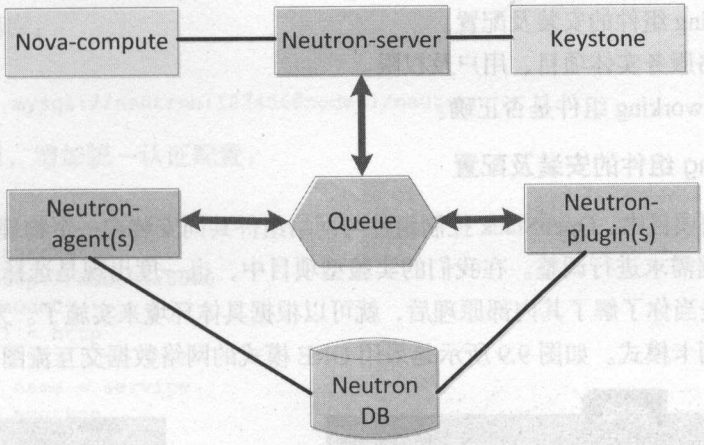


图 9.8　Networking 工作原理示意图

Networking 组件内部主要有三个部件组成。

- Neutron-server：接收用户的 RESTful API 请求和分发处理任务，比如 Nova-api 创建网络的请求。
- Neutron-plugin(s)：主要服务于 Neutron-server，运行在网络节点上（本书为控制节点），提供 RESTful API 作为访问 Neutron 的入口。
- Neutron-agent(s)：负责执行一些具体任务和操作，使用物理网络设备或虚拟化来完成实际的操作，比如实现路由操作的 L3 Agent。

此外，Neturon 还分别提供了二层（L2）交换及三层（L3）路由功能，对应于物理网络环境下的交换机和路由器实现。其具体实现的功能如下。

- Router：为租户提供路由功能及转发服务。
- Network：对应于一个真实物理网络中的二层网络（VLAN），从租户角度来看，它是私有的。
- Subnet：在网络中是三层的概念，给定一段 IPv4 或者 IPv6 地址和相关配置信息，使它附在一个二层网络上，指明这个网络上运行虚机的 IP 地址范围。

对于一个虚拟的二层网络结构来说，主要完成物理网卡和交换机的虚拟化，而在 Linux 环境下网络设备虚拟化主要有 TUN/VETH、Linux Bridge、OpenvSwitch 三种形式，本书使用 OpenvSwitch。

安装及配置 Networking 组件分为以下几步。

- 创建 Neutron 数据库及配置（略）。
- Networking 组件的安装及配置。
- 创建网络服务实体项目、用户及权限。
- 验证 Networking 组件是否正确。

1. Networking 组件的安装及配置

在一期、二期项目中，OpenStack 控制组件与网络组件共同安装在一个物理节点上，因此物理网络部分需要根据需求进行调整。在我们的实验型项目中，也一度出现是选择三网卡还是两网卡的困惑问题。但是当你了解了其内部原理后，就可以根据具体环境来实施了。为了管理上的方便，我们暂且采用两网卡模式。如图 9.9 所示是采用 GRE 模式的网络数据交互流图。

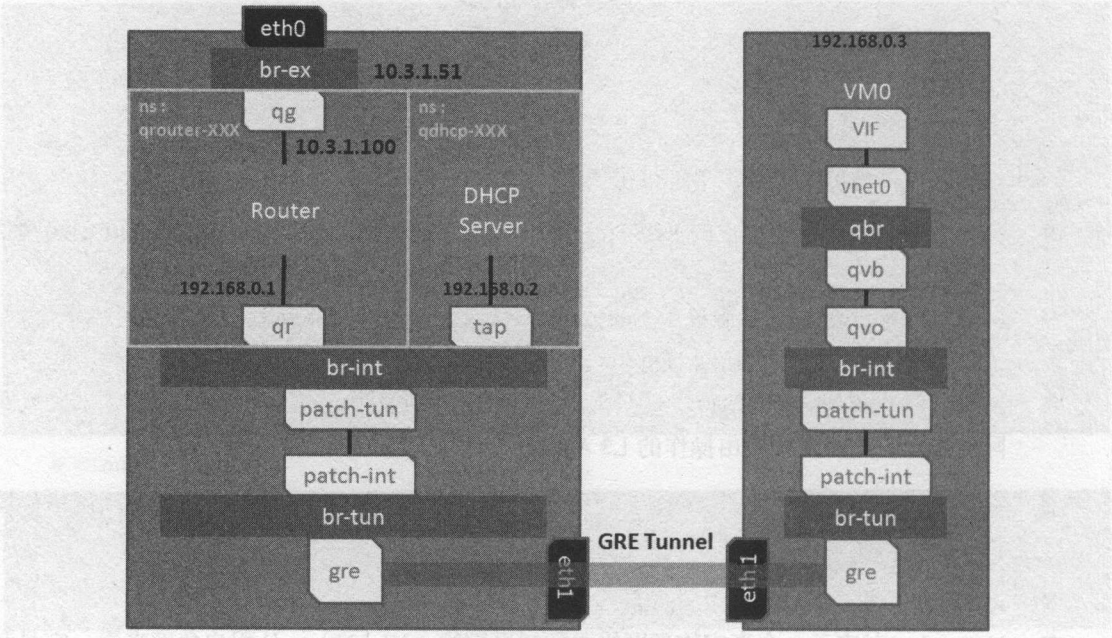


图 9.9　采用 GRE 模式的网络数据交互流图

在图 9.9 中，左边是控制节点的网络内部情况，右边是计算节点的网络。虚机之间的信息交互使用隧道模式，根据该图，我们分别对控制节点和计算节点进行组件的安装及配置。

（1）在控制节点上安装网络控制组件、网络组件并配置，命令如下。

① 安装网络控制组件：

```
# apt-get install neutron-server neutron-plugin-ml2 python-neutronclient
```

接下来配置网络参数，编辑/etc/neutron/neutron.conf 文件，添加或修改以下内容。下面的配置信息有些与镜像服务配置、计算服务配置一样。

修改连接数据库：

```
[database]
connection = mysql://neutron:123456@node01/neutron
```

对于一期项目，增加统一认证配置：

```
[default]
auth_strategy = keystone
[keystone_authtoken]
auth_uri = http://node01:5000
auth_host = node01
auth_protocol = http
auth_port = 35357
admin_tenant_name = service
admin_user = neutron
admin_password = 123456
```

增加消息队列配置：

```
[default]
rpc_backend = neutron.OpenStack.common.rpc.impl_kombu
rabbit_host = controller
rabbit_password = 123456
```

增加支持 ML2 插件、路由服务及 IP 地址重复使用机制：

```
[default]
core_plugin = ml2
service_plugins = router
allow_overlapping_ips = True
```

同时，增加网络组件与计算组件间信息交互配置：

```
[default]
notify_nova_on_port_status_changes = True
notify_nova_on_port_data_changes = True
nova_url = http://node01:8774/v2
nova_admin_username = nova
nova_admin_tenant_id = SERVICE_TENANT_ID
nova_admin_password = 123456
nova_admin_auth_url = http://node01:35357/v2.0
```

SERVICE_TENANT_ID 需要通过命令来获取，具体如下：

```
$ source admin.sh
$ keystone tenant-get service
```

然后把获取的 ID 替换成 SERVICE_TENANT_ID。

对于二期项目，增加统一认证配置：

```
[default]
auth_strategy = keystone
[keystone_authtoken]
auth_uri = http://node01:5000
auth_url = http://node01:35357
auth_plugin = password
project_domain_id = default
user_domain_id = default
project_name = service
username = neutron
password = 123456
```

增加消息队列配置：

```
[default]
verbose = True
rpc_backend = rabbit
[oslo_messaging_rabbit]
rabbit_host = node01
rabbit_userid = OpenStack
rabbit_password = 123456
```

增加支持 ML2 插件、路由服务及 IP 地址重复使用机制：

```
[default]
core_plugin = ml2
service_plugins = router
allow_overlapping_ips = True
```

同时，增加网络组件与计算组件间的信息交互配置：

```
[default]
notify_nova_on_port_status_changes = True
notify_nova_on_port_data_changes = True
nova_url = http://node01:8774/v2
[nova]
auth_url = http://node01:35357
auth_plugin = password
project_domain_id = default
user_domain_id = default
region_name = RegionOne
project_name = service
username = nova
password = 123456
```

在一期、二期项目中，ML2 插件文件/etc/neutron/plugins/ml2/ml2_conf.ini 的配置是一样的，只需要增加以下内容即可：

```
[ml2]
type_drivers = flat,vlan,gre,vxlan
```

```
tenant_network_types = gre
mechanism_drivers = openvswitch
[ml2_type_gre]
tunnel_id_ranges = 1:999
```

并增加安全配置信息：

```
[securitygroup]
enable_security_group = True
enable_ipset = True
firewall_driver =
neutron.agent.linux.iptables_firewall.OVSHybridIptablesFirewallDriver
```

②安装网络组件。

首先配置底层系统内核网络准入参数，编辑/etc/sysctl.conf 文件：

```
net.ipv4.ip_forward=1
net.ipv4.conf.all.rp_filter=0
net.ipv4.conf.default.rp_filter=0
```

查看添加的内核网络准入参数是否成功：

```
# sysctl -p | grep net.ipv4
```

```
root@node01:~# sysctl -p | grep net.ipv4
net.ipv4.conf.default.rp_filter = 0
net.ipv4.conf.all.rp_filter = 0
net.ipv4.ip_forward = 1
```

然后安装相应的网络组件，一期项目与二期项目唯一不同的是，一期项目增加安装了 openvswitch-datapath-dkms 包，二期项目增加安装了 neutron-metadata-agent 包，安装命令分别如下。

一期项目：

```
# apt-get install neutron-plugin-ml2 neutron-plugin-openvswitch-agent \
neutron-l3-agent neutron-dhcp-agent openvswitch-datapath-dkms
```

二期项目：

```
# apt-get install neutron-plugin-ml2 neutron-plugin-openvswitch-agent \
neutron-l3-agent neutron-dhcp-agentneutron-metadata-agent
```

因为控制服务与网络服务在同一节点上，在配置控制部分时已经做了部分工作。因此，这里我们只需要配置网络服务的相关参数即可。在 ML2 配置文件中增加相应的参数，编辑/etc/neutron/plugins/ml2/ml2_conf.ini 文件：

```
[ml2_type_flat]
flat_networks = external
```

配置隧道使用的网卡 IP 地址：

```
[ovs]
local_ip = 10.0.1.*
```

```
bridge_mappings = external:br-ex
[agent]
tunnel_types = gre
```

配置 L3 层代理信息，编辑/etc/neutron/l3_agent.ini 文件：

```
[default]
verbose = True
interface_driver = neutron.agent.linux.interface.OVSInterfaceDriver
external_network_bridge =  br-ex
router_delete_namespaces = True
```

配置 DHCP 代理信息，编辑/etc/neutron/dhcp_agent.ini 文件：

```
[default]
verbose = True
interface_driver = neutron.agent.linux.interface.OVSInterfaceDriver
dhcp_driver = neutron.agent.linux.dhcp.Dnsmasq
dhcp_delete_namespaces = True
dnsmasq_config_file = /etc/neutron/dnsmasq-neutron.conf
```

接下来，对 dnsmasq 的相关配置进行修改，设置 DHCP 的 MTU 值，编辑/etc/neutron/dnsmasq-neutron.conf 文件：

```
dhcp-option-force=26,1454
```

作为网络节点，需要配置 metadata 代理，编辑/etc/neutron/metadata_agent.ini 文件：

```
[default]
verbose = True
auth_uri = http://node01:5000
auth_url = http://node01:35357
auth_region = RegionOne
auth_plugin = password
project_domain_id = default
user_domain_id = default
project_name = service
username = neutron
password = 123456
nova_metadata_ip = node01
metadata_proxy_shared_secret = 123456
```

除了编辑 metadata 的配置文件外，还需要再次修改/etc/nova/nova.conf 文件：

```
[neutron]
service_metadata_proxy = True
metadata_proxy_shared_secret = 123456
```

最后，重启服务并添加外网路由网卡信息：

```
# service nova-api restart
# service openvswitch-switch restart
```

```
# ovs-vsctl add-br br-ex
# ovs-vsctl add-port br-ex eth0
```

注意：这里是通过 eth0 进行数据转发的，那么需要对 eth0 网卡的配置进行调整，才能适应网络的要求。调整内容如下：

```
auto br-ex
iface br-ex inet static
      address 10.3.1.*
      netmask 255.255.255.0
      network 10.3.1.0
      broadcast 10.3.1.255
      gateway 10.3.1.254
      dns-nameservers 114.114.114.114
auto eth0
iface eth0 inet manual
      up ifconfig $IFACE 0.0.0.0 up
      up ip link set $IFACE promisc on
      down ip link set $IFACE promisc off
      down ifconfig $IFACE down
```

上面的配置是把虚拟网卡 br-ex 嫁接到 eth0 上，而物理网卡 eth0 在这里只启动链路转发等功能。具体可参照图 9.9。

（2）在计算节点上安装网络的支撑组件。其在配置方面很大部分与控制节点相同，为了减少篇幅，直接将相关内容复制到配置文件中。

首先，配置底层系统内核网络准入参数，编辑/etc/sysctl.conf 文件：

```
net.ipv4.conf.all.rp_filter=0
net.ipv4.conf.default.rp_filter=0
net.bridge.bridge-nf-call-iptables=1
net.bridge.bridge-nf-call-ip6tables=1
```

然后，安装相应的网络支撑组件：

```
# apt-get install neutron-plugin-ml2 neutron-plugin-openvswitch-agent
```

接下来，编辑配置文件，关于一期、二期项目的相同文件名的不同配置内容，请参考上面的讲解。

在 neutron.conf 中添加[default]、[keystone_authtoken]的内容，在二期项目中还需要添加[oslo_messaging_rabbit]的内容。

在 ml2_conf.ini 中添加[ml2]、[ml2_type_gre]、[securitygroup]、[ovs]、[agent]的内容。

最后，配置 OpenvSwitch 服务及 Nova 服务：

```
# service openvswitch-switch restart
# ovs-vsctl add-br br-int
```

修改/etc/nova/nova.conf文件，添加以下内容。

一期项目：

```
[default]
network_api_class = nova.network.neutronv2.api.API
neutron_url = http://node01:9696
neutron_auth_strategy = keystone
neutron_admin_tenant_name = service
neutron_admin_username = neutron
neutron_admin_password = 123456
neutron_admin_auth_url = http://node01:35357/v2.0
linuxnet_interface_driver = nova.network.linux_net.LinuxOVSInterfaceDriver
firewall_driver = nova.virt.firewall.NoopFirewallDriver
security_group_api = neutron
```

二期项目：

```
[default]
network_api_class = nova.network.neutronv2.api.API
security_group_api = neutron
linuxnet_interface_driver = nova.network.linux_net.LinuxOVSInterfaceDriver
firewall_driver = nova.virt.firewall.NoopFirewallDriver
[neutron]
url = http://node01:9696
auth_strategy = keystone
admin_auth_url = http://node01:35357/v2.0
admin_tenant_name = service
admin_username = neutron
admin_password = 123456
```

配置完成后，重启相应的服务：

```
# service nova-compute restart
# service neutron-plugin-openvswitch-agent restart
```

2. 创建网络服务实体项目、用户及权限

在控制节点上创建网络存储服务实体项目、用户及权限的过程与其他组件相同。

首先，导入管理员访问资源权限：

```
$ source admin.sh
```

接下来，一期项目，分别创建实体、用户及分配权限和连接点：

```
$ keystone user-create --name=neutron--pass=123456--email=neutron@126.com
$ keystone user-role-add --user=neutron --tenant=service --role=admin
$ keystone service-create --name=neutron --type=network\
--description="OpenStackNetworking"
$ keystone endpoint-create \
```

```
--service-id $(keystone service-list | awk '/ network / {print $2}') \
--publicurl http://node01:9696 --adminurl http://node01:9696 \
--internalurl http://node01:9696
```

二期项目，操作如下：

```
$ OpenStack user create --password-prompt neutron
$ OpenStack role add --project service --user neutron admin
$ OpenStack service create --name neutron-description "OpenStack\
 Networking" network
$ OpenStack endpoint create --publicurl http://node01:9696 \
--adminurl http://node01:9696 --internalurl http://node01:9696 \
--region RegionOne network
```

同步数据、删除临时数据库及重启服务命令如下：

```
# su -s /bin/sh -c "neutron-db-manage --config-file /etc/neutron/neutron.conf\
--config-file /etc/neutron/plugins/ml2/ml2_conf.iniupgrade head" neutron
# service nova-api restart
# service nova-scheduler restart
# service nova-conductor restart
# service neutron-server restart
```

3. 验证 Networking 组件是否正确

无论在控制节点还是计算节点上，都可以运行以下命令来验证服务是否正确。为了保证配置有效，在控制节点上需要先重启相应的服务：

```
# service neutron-plugin-openvswitch-agent restart
# service neutron-l3-agent restart
# service neutron-dhcp-agent restart
# service neutron-metadata-agent restart
```

然后，导入管理员认证信息后再验证：

```
$ source admin.sh
$ neutron agent-list
```

```
trusty@node01:~$ neutron agent-list
+--------------------------------------+--------------------+--------+-------+----------------+---------------------------+
| id                                   | agent_type         | host   | alive | admin_state_up | binary                    |
+--------------------------------------+--------------------+--------+-------+----------------+---------------------------+
| 0726fa74-69a2-44f5-8282-a2a1311c0707 | Metadata agent     | node01 | :-)   | True           | neutron-metadata-agent    |
| 431fe64b-15cf-4277-b43f-a2acfc26c1a1 | Loadbalancer agent | node01 | :-)   | True           | neutron-lbaas-agent       |
| 4ed25d7a-c09a-40d3-add3-fe0017f00e68 | Open vSwitch agent | node04 | :-)   | True           | neutron-openvswitch-agent |
| 67f558e7-7f64-4c28-8ab1-d3d96c24ce5f | Open vSwitch agent | node01 | :-)   | True           | neutron-openvswitch-agent |
| 7d6d0e1f-5037-4baf-a99d-8975f0600338 | Open vSwitch agent | node02 | :-)   | True           | neutron-openvswitch-agent |
| 7ddd81fb-a4ef-4253-b857-e3533795d47e | L3 agent           | node01 | :-)   | True           | neutron-l3-agent          |
```

4. 创建初始化网络

通过上面的测试表明网络组件安装成功。在正式创建虚机前，还需要初始化网络，包括外网和租户虚拟内网。

（1）外网初始化

首先，导入管理员访问权限：

```
$ source admin.sh
```

对于一期项目，创建外网，使用网络类型为 gre：

```
$ neutron net-create ext-net --shared --router:external=True
```

创建外网的一个子网：

```
$ neutron subnet-create ext-net --name ext-subnet --allocation-pool\
 start=10.3.1.52,end=10.3.1.200 --disable-dhcp --gateway \
10.3.1.254 10.3.1.0/24
```

对于二期项目，创建外网，使用网络类型为 flat：

```
$ neutron net-create ext-net --router:external \
--provider:physical_network external --provider:network_type flat
```

创建外网的一个子网：

```
$ neutron subnet-create ext-net 10.3.1.0/24 --name ext-subnet \
--allocation-pool start=10.3.1.52,end=10.3.1.200 \
--disable-dhcp --gateway 10.3.1.254
```

（2）租户内网初始化

首先，导入普通用户权限：

```
$ source test.sh
```

对于一期项目，创建租户网络：

```
$ neutron net-create test-net
```

创建租户的一个子网：

```
$ neutron subnet-create test-net --name test-subnet \
--gateway 192.168.1.1 192.168.1.0/24
```

对于二期项目，创建租户网络：

```
$ neutron net-create test-net
```

创建租户的一个子网：

```
$ neutron subnet-create test-net 192.168.1.0/24 \
--name test-subnet --gateway 192.168.1.1
```

最后，在一期、二期项目中创建租户虚拟路由，使得租户虚拟网络与外网连通：

```
$ neutron router-create test-router
$ neutron router-interface-add test-router test-subnet
$ neutron router-gateway-set test-router ext-net
```

（3）验证配置的正确性

在控制节点或者任意计算节点上，ping 租户路由网关，检验网络是否连通：

```
$ ping -c 3 10.3.1.52
```

```
trusty@node01:~$ ping -c 3 10.3.1.52
PING 10.3.1.52 (10.3.1.52) 56(84) bytes of data.
64 bytes from 10.3.1.52: icmp_seq=1 ttl=64 time=0.226 ms
64 bytes from 10.3.1.52: icmp_seq=2 ttl=64 time=0.276 ms
64 bytes from 10.3.1.52: icmp_seq=3 ttl=64 time=0.239 ms

--- 10.3.1.52 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1998ms
rtt min/avg/max/mdev = 0.226/0.247/0.276/0.021 ms
```

9.2.7　Ceph 分布式存储系统

对于以基础架构即服务部署和设计云计算来说，数据复制和存储是为了确保给用户提供完整性和服务连续性的实际前提条件。OpenStack 是一个云计算平台，那么数据复制和存储的持久化可以有多种存储系统，如 LVM、GlusterFS、Ceph、Sheepdog（关于它们的区别，可以在网上搜索查询）。本书重点讲解在 OpenStack 整合上发展较快的 Ceph 分布式存储系统。

Ceph 是一个符合 POSIX （Portable Operating System for UNIX）、开源的分布式存储系统，依据 GNU LGPL 而运行。该项目最初由 Sage Weill 于 2007 年开发，其理念是提出一个没有任何单点故障的集群，确保能够跨集群节点进行永久性数据复制。与在任何经典的分布式文件系统中一样，放入集群中的文件是条带化的，依据一种称为 Ceph CRUSH（Controlled Replication Under Scalable Hashing）的伪随机数据分布算法放入集群节点中。

Ceph 内部结构示意图如图 9.10 所示。

Ceph 内部结构主要有 4 个部分组成。

（1）Ceph 客户端

在 Linux 系统上显示文件系统用到一个公用的界面，该界面对用户是透明的，不同用户访问这个界面会得到不同的存储模式。但他们唯一看到的就是大量的存储空间，而不知道其下聚合的是大容量的存储池。用户通过这个界面只需要操作权限内的事情就可以了，比如扩充空间、备份文件等。

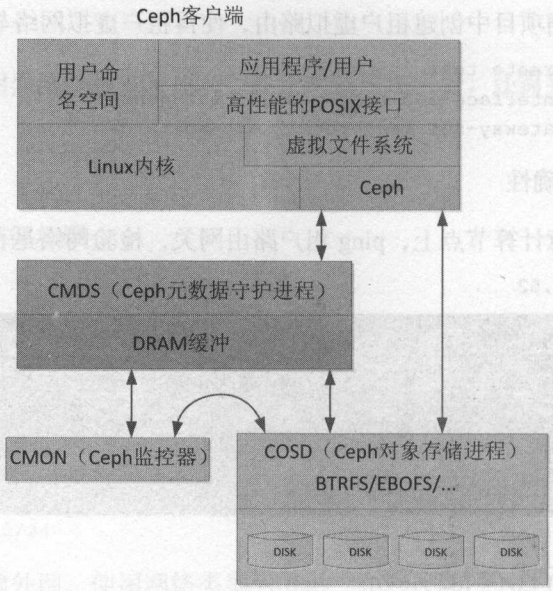


图 9.10　Ceph 内部结构示意图

（2）Ceph 元数据服务器

Ceph 元数据服务器用来管理文件系统的命名空间。虽然元数据与用户数据共同存放在存储集群上，但是对于两者的管理，需分别进行标记性管理，且支持可扩展性。

（3）Ceph 监控器

顾名思义，Ceph 监控器是在故障管理上 Ceph 自身修复的一种机制。当对象存储设备发生故障或添加新设备时，监控器会感知这些信息，然后做出相应的处理。

（4）Ceph 对象存储

Ceph 对象存储与传统的存储有些不同，它除了存储功能还有智能型功能，即支持与其他对象存储设备之间的通信与合作。

如图 9.11 所示是 Ceph 存储组件关系图，分布式对象存储系统 RADOS 是 Ceph 的核心，由自修复、自管理、智能的存储节点组成，它由上面提到的 CRUSH 算法来管理上千个存储节点，并把数据持久化到物理磁盘上。

关于 Ceph 更详细的介绍和内部流程，请参考官方文档。在我们的环境中，只有二期项目使用 Ceph 作为后端存储，因此，下面针对安装、部署、测试和与 OpenStack 整合方面的讲解只针对二期云平台。

图 9.11　Ceph 存储组件关系图

1. Ceph 集群的安装及配置

在安装 Ceph 时，一般有两种方式：一是通过源码编译安装；二是通过安装包安装。通过源码编译安装较为复杂，如果你需要深入了解 Linux 下源码编译等内容，使用该方法较为合适。为了部署的便捷性，我们采用官方安装包部署安装。

在二期项目中，将 Ceph 集群部署在计算节点上，而将它的部署控制部分放在了控制节点上，这样易于增加集群进行管理。另外，无密码登录和时间同步功能在 Ceph 存储集群中同样重要，必须先完成这些配置工作。由于将 Ceph 集群部署在计算节点上，而这些工作在前面章节中已经完成，所以这里不再赘述。

（1）在所有节点上添加 Ceph 库

```
# wget -q -O- 'https://download.ceph.com/keys/release.asc' |sudo apt-key add-
# echo deb http://download.ceph.com/debian-hammer/ $(lsb_release -sc) main| \
sudo tee /etc/apt/sources.list.d/ceph.list
```

在控制节点上更新系统并安装 Ceph 部署工具包：

```
$ sudo apt-get update && sudo apt-get install ceph-deploy
```

在当前用户目录下新建 my-cluster 文件夹，并创建控制节点为部署主机：

```
$ mkdir my-cluster
$ cd my-cluster
$ ceph-deploy new node01
```

配置该文件夹中的 ceph.conf 文件，设置默认池为 3 个，存储网络为 10.0.1.*，添加信息如下：

```
[global]
Osd pool default size = 3
Public network = 10.0.1.*/24
```

接下来，初始化 Ceph 监控器（监控器可以有多个，但需要做 HA。这里采用单监控器），并修改 Ceph 的认证文件权限：

```
$ ceph-deploy mon create-initial
$ chmod +r ceph.client.admin.keyring
```

（2）将 Ceph 系统部署到计算节点上

首先，将 Ceph 相关包安装到计算节点上，并检查计算节点的磁盘情况：

```
$ ceph-deploy install node02 node03 node04
$ ceph-deploy disk list node02 node03 node04
```

然后，确定磁盘、格式化磁盘及创建 OSD：

```
$ ceph-deploy disk  zap node02:sd{b,c}  node03:sd{b,c} node04:sd{b,c}
$ ceph-deploy osd create node02:sd{b,c}  node03:sd{b,c} node04:sd{b,c}
```

把相关 Ceph 配置文件复制到计算节点上，并创建元数据服务器：

```
$ ceph-deploy admin node02 node03 node04
$ ceph-deploy mds  create node01
```

（3）重新部署 Ceph

在计算节点上重新部署 Ceph，要做的事情比较多，大概总结如下。

① 停止所有的 Ceph 进程：

```
$ sudo stop ceph-all
```

② 卸载所有的 Ceph 程序：

```
$ ceph-deploy uninstall node02 node03 node04
```

③ 删除 Ceph 相关包及包数据文件：

```
$ ceph-deploy purge node02 node03 node04
$ ceph-deploy purgedata node02 node03 node04
```

④ 删除 Ceph 锁文件：

```
$ ceph-deploy forgetkeys
```

然后，再按照上面的方式部署 Ceph 系统。

2. OpenStack 整合 Ceph 存储系统

在官方给出的 OpenStack 配置中，Glance 组件使用 Swift 对象存储，Cinder 组件使用 LVM 逻辑卷，虚机使用本地磁盘。那么，就会产生一个问题：在创建虚机时，从 Swift 中读取镜像文件，然后部署到 LVM 中。这样不仅在速度上造成问题，而且占用更多的网络链路资源，影响性能。所以，业界普遍对 Glance、Cinder、VM 磁盘使用统一化存储，便于管理和提升性能。

Ceph 就是这样的一个统一化存储系统，它支持对象存储、块存储和文件存储，因此，使用它可以把 Glance 镜像文件、Cinder 后端存储、VM-Backup 全部集中放到该存储系统中，即使有数据复制、数据备份及其他功能的使用，也只是在分布式存储系统中进行，不会占用更多的带宽及影响云平台性能，而且还便于管理和维护。

OpenStack 与 Ceph 的结合拥有诸多好处，而这里只是提到部分功能。那么，如何把它们整合在一起，以及如何实现整合呢？首先，由 OpenStack 与 Ceph 内部连接关系图（见图 9.12）能了解二者之间的关系。

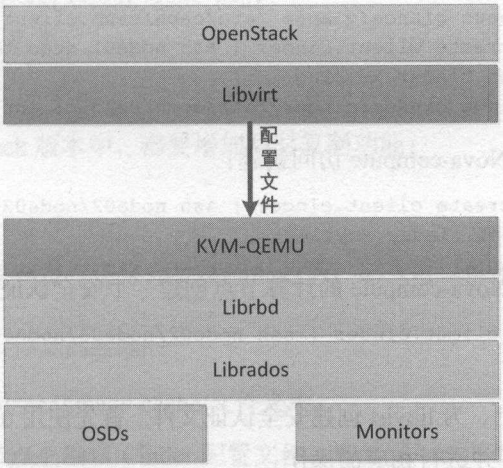


图 9.12　OpenStack 与 Ceph 内部连接关系图

接下来，我们通过 Ceph 与 OpenStack 组件配置来进一步了解二者之间的关系。大家需要注意：虚机使用 Ceph 作为存储时，磁盘必须是 RAW 格式的，而不能是 QCOW2 格式的。

（1）针对 OpenStack 组件，进行 Ceph 相应包的安装和配置

首先，分别创建卷、镜像、虚机使用的资源池：

```
$ceph osd pool create volumes 128
$ceph osd pool create images 128
$ceph osd pool create vms 128
```

在运行 glance-api 的节点（控制节点）上，安装 rbd 连接库：

```
# apt-get install python-rbd
```

在计算节点（存储节点）上，安装 Ceph 通用组件：

```
# apt-get install ceph-common
```

接下来，如果在构建分布式存储时使用了认证，那么还需要在任一 Ceph 节点上为 Glance、

Cinder、Nova 创建认证用户：

```
$ ceph auth get-or-create client.cinder mon 'allow r' osd 'allow \
class-read object_prefix rbd_children, allow rwx pool=volumes, allow \
rwx pool=vms, allow rx pool=images'
$ ceph auth get-or-create client.glance mon 'allow r' osd 'allow \
class-read object_prefix rbd_children, allow rwx pool=images'
```

同时，在控制节点上，为 client.cinder、clent.glance 创建访问钥匙：

```
$ ceph auth get-or-create client.glance | ssh node01 sudo tee \
/etc/ceph/ceph.client.glance.keyring
$ ssh node01 sudo chown glance:glance /etc/ceph/ceph.client.glance.keyring
$ ceph auth get-or-create client.cinder | ssh node01 sudo tee \
/etc/ceph/ceph.client.cinder.keyring
$ ssh node01 sudo chown cinder:cinder /etc/ceph/ceph.client.cinder.keyring
```

在计算节点上，创建 Nova-compute 访问钥匙：

```
$ ceph auth get-or-create client.cinder | ssh node02/node03/node04 sudo tee\
 /etc/ceph/ceph.client.cinder.keyring
```

另外，还需要给运行 Nova-compute 的计算节点创建一个安全认证的副本：

```
$ ceph auth get-key client.cinder | ssh node02/node03/node04 tee \
client.cinder.key
```

接下来，在计算节点上，为 libvirt 创建安全认证文件。首先使用 uuidgen 获取一个随机数，然后写入一个 XML 文件中，并进行相应的操作：

```
uuidgen
d20d433d-952c-4fc7-a3af-63235a5551c5
cat > secret.xml <<EOF
<secret ephemeral='no' private='no'>
  <uuid>d20d433d-952c-4fc7-a3af-63235a5551c5</uuid>
  <usage type='ceph'>
    <name>client.cinder secret</name>
  </usage>
</secret>
EOF

sudo virsh secret-define --file secret.xml
Secret d20d433d-952c-4fc7-a3af-63235a5551c5 created
sudo virsh secret-set-value --secret d20d433d-952c-4fc7-a3af-63235a5551c5 \
 --base64 $(cat client.cinder.key) && rm client.cinder.key secret.xml
```

注意：uuidgen 只使用一次，也就是只使用一个序列号，因为在稍后配置 Nova 组件时会用到。

（2）OpenStack 组件配置

① 在控制节点上对 Glance 文件进行配置，编辑/etc/glance/glance-api.conf 文件：

```
[default]
default_store = rbd
  [glance_store]
stores = rbd
stores = glance.store.rbd.Store
rbd_store_pool = images
rbd_store_user = glance
rbd_store_ceph_conf = /etc/ceph/ceph.conf
rbd_store_chunk_size = 8
rados_connect_timeout = -1
show_image_direct_url = True
```

同时，在任何 OpenStack 版本中，都要增加写时复制功能：

```
[default]
show_image_direct_url = True
```

如果在某一节点上需要有缓存镜像而提高速度的话，那么可以修改下面的参数：

```
[paste_deploy]
flavor = keystone+cachemanagement
```

镜像被缓存到/var/lib/glance/image-cache/文件夹中。

② 在运行 Cinder 的节点上编辑 Cinder 配置文件，即编辑/etc/cinder/cinder.conf 文件：

```
[default]
enabled_backends = rbd
[rbd]
volume_driver = cinder.volume.drivers.rbd.RBDDriver
rbd_pool = volumes
rbd_ceph_conf = /etc/ceph/ceph.conf
rbd_flatten_volume_from_snapshot = false
rbd_max_clone_depth = 5
rbd_store_chunk_size = 8
rados_connect_timeout = -1
glance_api_version = 2
```

以及配置 Nova 使用 Ceph 块设备：

```
rbd_user = cinder
rbd_secret_uuid = d20d433d-952c-4fc7-a3af-63235a5551c5
```

③ 为了在虚机启动时，在 Ceph 分布式存储系统上引导加载磁盘，我们需要对计算节点上的 Nova 配置文件做相应修改，即编辑/etc/nova/nova.conf 文件，添加如下内容：

```
[client]
rbd cache = true
```

```
rbd cache writethrough until flush = true
admin socket = /var/run/ceph/$cluster-$type.$id.$pid.$cctid.asok
```

同时，增加虚机运行资源池配置：

```
[libvirt]
images_type = rbd
images_rbd_pool = vms
images_rbd_ceph_conf = /etc/ceph/ceph.conf
rbd_user = cinder
rbd_secret_uuid =  d20d433d-952c-4fc7-a3af-63235a5551c5
```

然后，重启所有的服务：

```
# glance-control api restart
# service nova-compute restart
# service cinder-volume restart
```

至此，OpenStack 整合 Ceph 分布式存储系统配置完毕。

3. 验证 Ceph 和虚机运行是否正确

在上面的章节中，我们对 Ceph 的安装，以及与 OpenStack 的整合进行了详细配置。那么，是否正确？需要验证才能做出判断。

（1）Ceph 集群服务正确性验证

检查 Ceph 服务有多个命令，下面通过命令来展示它的正确情况：

```
# ceph status
```

```
root@node01:~# ceph status
    cluster ea9c67a7-02a6-4518-95e4-5506016101ea
     health HEALTH_OK
     monmap e1: 1 mons at {node01=222.249.249.12:6789/0}
            election epoch 1, quorum 0 node01
     osdmap e28013: 6 osds: 6 up, 6 in
      pgmap v12260448: 576 pgs, 5 pools, 770 GB data, 165 kobjects
            1553 GB used, 15175 GB / 16728 GB avail
                 576 active+clean
  client io 91352 B/s wr, 23 op/s
```

在该命令输出的信息中，我们很容易看出该集群是健康的、正确的。此外，还显示了存储容量、使用情况及 I/O 读写速度等。

（2）OpenStack 组件使用存储验证

我们从 Glance、Cinder、Nova 三个方面进行说明。先导入管理用户权限：

```
$ source admin.sh
```

① Glance 验证：

```
$ glance image-create --name "cirros-0.3.3-x86_64" --file ./cirros.img \
```

```
--disk-format raw --container-format bare --visibility public -progress
```

Property	Value
checksum	133eae9fb1c98f45894a4e60d8736619
container_format	bare
created_at	2015-08-12T02:12:55Z
disk_format	raw
id	1c56883b-b242-4d1d-9f4b-843d6cff82c3
min_disk	0
min_ram	0
name	cirros-0.3.3-x86_64
owner	2d781a3dbb6d4da29b06622a4a6cf892
protected	False
size	13200896
status	active
tags	[]
updated_at	2015-08-12T02:12:57Z
virtual_size	None
visibility	public

然后，使用“glance image-list”命令检查是否上传成功：

```
trusty@node01:~$ glance image-list
```

ID	Name
1c56883b-b242-4d1d-9f4b-843d6cff82c3	cirros-0.3.3-x86_64

② Cinder 验证：

```
$ cinder create --image-id 1c56883b-b242-4d1d-9f4b-843d6cff82c3 \
 --display-name volume1 2
```

```
trusty@node01:~$ cinder create --image-id 1c56883b-b242-4d1d-9f4b-843d6cff82c3 --display-name volume1 2
```

Property	Value
attachments	[]
availability_zone	nova
bootable	false
consistencygroup_id	None
created_at	2016-03-26T10:29:03.000000
description	None
encrypted	False
id	4d77c799-48a0-4f9b-b7c9-e0a70a63f661
metadata	{}
multiattach	False
name	volume1
os-vol-host-attr:host	node02@rbd#RBD_iSCSI
os-vol-mig-status-attr:migstat	None
os-vol-mig-status-attr:name_id	None
os-vol-tenant-attr:tenant_id	2d781a3dbb6d4da29b06622a4a6cf892
os-volume-replication:driver_data	None
os-volume-replication:extended_status	None
replication_status	disabled
size	2
snapshot_id	None
source_volid	None
status	creating
user_id	5a45a393d38749faa0217886c9f0855f
volume_type	None

其中，矩形框中的数据说明卷被创建在 node02 上，而且使用的是 node02 的分布式存储空间。

④ Nova 验证：在 Nova 验证中，需要直接加载虚机模板，然后查看虚机存储空间的变化情况。前后分别使用两次“ceph df”命令，来查看存储空间的变化情况。

在 Web 页面上，创建一个虚机实例，如图 9.13 所示。

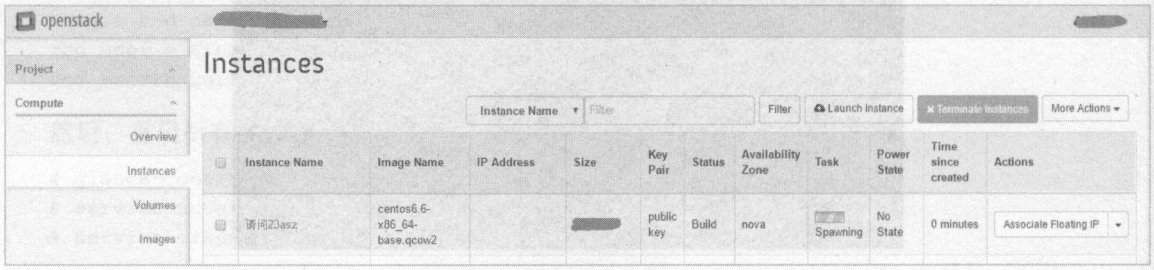


图 9.13　创建虚机实例

创建前后的存储使用情况如下：

```
root@node01:~# ceph df
GLOBAL:
    SIZE       AVAIL      RAW USED     %RAW USED
    16728G     15175G        1553G          9.29
POOLS:
    NAME        ID     USED      %USED     MAX AVAIL     OBJECTS
    rbd         0          0         0          7543G           0
    volumes     1       334G      2.00          7543G       89057
    images      2       243G      1.45          7543G       31157
    backups     3          0         0          7543G           0
    vms         4       192G      1.15          7543G       49361
root@node01:~# ceph df
GLOBAL:
    SIZE       AVAIL      RAW USED     %RAW USED
    16728G     15172G        1556G          9.30
POOLS:
    NAME        ID     USED      %USED     MAX AVAIL     OBJECTS
    rbd         0          0         0          7541G           0
    volumes     1       334G      2.00          7541G       89057
    images      2       243G      1.45          7541G       31157
    backups     3          0         0          7541G           0
    vms         4       193G      1.16          7541G       49743
```

上图说明，使用 Ceph 存储空间进行了虚机磁盘加载。

9.2.8　Dashboard（Horizon）组件

在 OpenStack 组件中，我们创建、管理虚机或者进行其他操作，可以使用客户端命令，但对于大型的云平台管理来说不灵活、不方便。因此，Horizon 组件应运产生，给用户提供一个简单、直观的操作界面，且易于管理各种云平台上的服务。

1. Dashboard 组件的安装及配置

在控制节点上，安装及配置 Horizon 相对于其他组件要简单得多，具体操作如下。

在一期项目中，没有提前用到 Apache 服务，因此前面没有安装 Web 服务。下面统一安装 Apache 服务及 Horizon 服务：

```
# apt-get install apache2 memcached libapache2-mod-wsgi OpenStackdashboard
```

如果不需要 Ubuntu 风格的 OpenStack Horizon 界面，则可以卸载该包文件：

```
# apt-get remove --purge OpenStack-dashboard-ubuntu-theme
```

在二期项目中，安装 Horizon 的 Web 组件：

```
# apt-get install OpenStack-dashboard
```

在一期、二期项目中，对配置文件的操作相同。下面编辑配置文件。

编辑 Horizon 的 Web 配置文件/etc/OpenStack-dashboard/local_settings.py：

```
OPENSTACK_HOST = "node01"
ALLOWED_HOSTS = '*'
OPENSTACK_KEYSTONE_DEFAULT_ROLE = "user"
TIME_ZONE = "Asia/Shanghai"
```

为了使用户访问时进行加速，需要增加缓存信息：

```
CACHES = {
    'default': {
        'BACKEND': 'django.core.cache.backends.memcached.
        MemcachedCache',
        'LOCATION': '127.0.0.1:11211',
    }
}
```

最后，重启 Web 服务：

```
# service apache2 reload
```

在一期项目中，还需要重启内存缓存服务：

```
# service memcached restart
```

2. 验证操作页面组件是否正确

相应的服务重启后，就可以通过 URL 来访问云平台，并进行虚机创建或其他操作了，如图 9.14 所示。

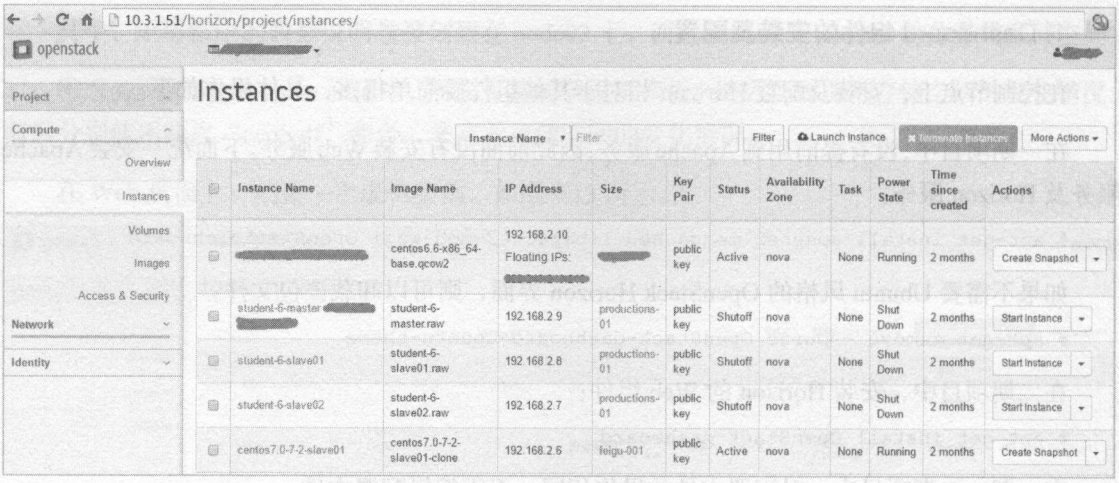


图 9.14　云平台管理页面

9.3　Identity（Keystone）与 LDAP 的整合

OpenStack 作为私有云和公有云的开源软件，诸多组件的权限管理全部交给了 Keystone 来处理。在多数企业内部，在构建私有云之前，已经运行着很多内部系统。那么，已有系统与云平台之间想必会产生账号兼容问题。为了使已有账号系统在云平台下也产生作用，同时便于账号管理人员管理，我们可以采用 AD（活动目录）或 LDAP（轻量级目录服务）来实现。这种实现就是为了解决单一用户账号登录系统问题。

在我们的环境中，采用 LDAP 来实现该功能。

（1）将 LDAP 安装到系统中

```
# apt-get install ldap-utils
# apt-get install slapd
```

验证 LDAP 登录，查看/etc/hosts：

```
127.0.0.1       localhost
10.3.1.51  node01 node01.feiguyun.com
10.3.1.52  node02 node02.feiguyun.com
```

由上图可以看出本环境的登录 DN 为：cn=admin，dc=feiguyun，dc=com。使用如下命令来验证配置的正确性：

```
# ldapsearch -x -LLL -Hldap:/// -b dc=feiguyun,dc=com dn
```

LDAP 的默认 Schema 是不能与 OpenStack 配合使用的，某些属性在默认 Schema 里还没有，需要进行修改并添加存储 OpenStack 相关模型，以便保存数据。在使用的代码中包括 modify.ldif

和 add.ldif。

modify.ldif 的内容如下：

```
    dn: cn={0}core,cn=schema,cn=config
    changetype: modify
    add: olcAttributeTypes
    olcAttributeTypes: {52}( 2.5.4.66 NAME 'enabled' DESC 'RFC2256: enabled of a group'
EQUALITY booleanMatch SYNTAX 1.3.6.1.4.1.1466.115.121.1.7 SINGLE-VALUE )
    dn: cn={0}core,cn=schema,cn=config
    changetype: modify
    delete: olcObjectClasses
    olcObjectClasses: {7}( 2.5.6.9 NAME 'groupOfNames' DESC 'RFC2256: a group of names
(DNs)' SUP top STRUCTURAL MUST ( member $ cn ) MAY ( businessCategory $ seeAlso $ owner
$ ou $ o $ description ) )
    -
    add: olcObjectClasses
    olcObjectClasses: {7}( 2.5.6.9 NAME 'groupOfNames' DESC 'RFC2256: a group of names
(DNs)' SUP top STRUCTURAL MUST ( member $ cn ) MAY ( businessCategory $ seeAlso $ owner
$ ou $ o $ description $ enabled) )
    dn: cn={3}inetorgperson,cn=schema,cn=config
    changetype: modify
    delete: olcObjectClasses
    olcObjectClasses: {0}( 2.16.840.1.113730.3.2.2 NAME 'inetOrgPerson' DESC 'RFC2798:
Internet Organizational Person' SUP organizationalPerson STRUCTURAL MAY ( audio
$ businessCategory $ carLicense $ departmentNumber $ displayName $ employeeNumber
$ employeeType $ givenName $ homePhone $ homePostalAddress $ initials $ jpegPhoto
$ labeledURI $ mail $ manager $ mobile $ o $ pager $ photo $ roomNumber $ secretary $ uid
$ userCertificate $ x500uniqueIdentifier $ preferredLanguage $ userSMIMECertificate
$ userPKCS12 ) )
    -
    add: olcObjectClasses
    olcObjectClasses: {0}( 2.16.840.1.113730.3.2.2 NAME 'inetOrgPerson' DESC 'RFC2798:
Internet Organizational Person' SUP organizationalPerson STRUCTURAL MAY ( audio
$ businessCategory $ carLicense $ departmentNumber $ displayName $ employeeNumber
$ employeeType $ givenName $ homePhone $ homePostalAddress $ initials $ jpegPhoto
$ labeledURI $ mail $ manager $ mobile $ o $ pager $ photo $ roomNumber $ secretary $ uid
$ userCertificate $ x500uniqueIdentifier $ preferredLanguage $ userSMIMECertificate
$ userPKCS12 $ description $ enabled $ email ) )
```

add.ldif 的内容如下：

```
dn: ou=users,dc=feiguyun,dc=com
objectClass: top
objectClass: comanizationalUnit
dn: ou=projects,dc=feiguyun,dc=com
objectClass: top
objectClass: comanizationalUnit
dn: ou=roles,dc=feiguyun,dc=com
```

```
objectClass: top
objectClass: comanizationalUnit
dn: ou=groups,dc=feiguyun,dc=com
objectClass: top
objectClass: comanizationalUnit
dn: ou=domains,dc=feiguyun,dc=com
objectClass: top
objectClass: comanizationalUnit
```

添加相关内容后，执行如下命令：

```
# ldapmodify -c -Y EXTERNAL -H ldapi:/// -f modify.ldif
# ldapadd -x -c -D"cn=admin,dc=feiguyun,dc=com" -w "123456" -f add.ldif
```

注：123456 是安装 LDAP 时的 root 密码。

（2）配置 Keystone 内容并初始化用户信息

将 Keystone 的后端配置成 LDAP，修改/etc/keystone/keystone.conf 文件：

```
[identity]
driver = keystone.identity.backends.ldap.Identity
```

同时，添加 LDAP 方面的配置信息：

```
[ldap]
url = ldap://10.3.1.52
user = cn=admin,dc=feiguyun,dc=com
password = 123456
suffix = dc=feiguyun,dc=com
use_dumb_member = True
allow_subtree_delete = False

user_tree_dn = ou=users,dc=feiguyun,dc=com
tenant_tree_dn = ou=projects,dc=feiguyun,dc=com
role_tree_dn = ou=roles,dc=feiguyun,dc=com
group_tree_dn = ou=groups,dc=feiguyun,dc=com
domain_tree_dn = ou=domains,dc=feiguyun,dc=com
```

然后，重启 Keystone 服务：

```
# service apache2 reload
```

最后，初始化 Keystone 基本用户。

这一步，国外大牛们已经写了相应的脚本，从网上可以搜到。这部分代码也如上面一样，提供代码包 OpenStack_basic.sh，并执行如下命令：

```
# sh OpenStack_basic.sh
```

（3）验证 Keystone 是否正确

上述配置结束后，我们就可以验证 Keystone 是否正确了。分别执行以下命令进行检查：

```
$ admin.sh
$ OpenStack user list
$ OpenStack rolelist
$ OpenStacktenant list
$ OpenStackuser role list
```

最后，重启 OpenStack 所有的进程，再验证 Nova、Cinder、Glance 情况：

```
$ nova list
$ cinder list
$ glance image-list
```

9.4　配置 Image 组件大镜像部署

当我们使用上面的配置部署 OpenStack 系统后，运行一般的虚机系统是没什么问题的。但接下来要讲的是：如果你拥有一个大镜像文件，要传送到 Glance 镜像中来部署虚机，该怎么办？

首先，我们使用的 Glance API 是 v2 或者 v3 版本。无论 v2 还是 v3 版本，使用 Glance 上传镜像都会存在中断问题。要解决这个问题，就要从配置文件入手，需要修改的配置文件有 glance-api.conf 和 glance-cache.conf。

编辑/etc/glance/glance-api.conf 文件：

```
[default]
image_size_cap = 1099511627776（镜像默认 1TB 大小）
data_api = glance.db.registry.api
[rbd]
hw_scsi_model=virtio-scsi
hw_disk_bus=scsi
hw_qemu_guest_agent=yes
os_require_quiesce=yes
```

编辑/etc/glance/ glance-cache.conf 文件：

```
[default]
image_cache_max_size = 64424509400（调整默认大小）
```

然后，测试上传超过 50GB 的大镜像文件，很容易上传完成，而不会中断。

```
trusty@node03:~/imgbak$ ll -h
total 214G
drwxrwxr-x 2 trusty trusty 4.0K Jan 11 11:16  /
drwx------ 7 trusty trusty 4.0K Mar 15 21:50  /
-rw-r--r-- 1 trusty trusty 1.4G Jan  8 17:21 centos6.6-x86_64.qcow2
-rw-r--r-- 1 trusty trusty 1.2G Jan  8 17:23 centos7.0-x86_64.qcow2
-rw-r--r-- 1 trusty trusty  50G Jan  5 23:52 student-6-master.raw
-rw-r--r-- 1 trusty trusty  50G Jan  5 23:22 student-6-salve01.raw
-rw-r--r-- 1 trusty trusty  50G Jan  5 23:42 student-6-salve02.raw
```

将大镜像文件上传到 Ceph 存储系统中：

```
$ glance image-create --name test --file  ./imgbak/student-6-master.raw --disk-format
raw --container-format bare --visibility public -progress
```

```
trusty@node03:~$ glance  image-list
+--------------------------------------+--------------------------------------+
| ID                                   | Name                                 |
+--------------------------------------+--------------------------------------+
| 5fbff436-3bfa-4666-a569-d7d31e677995 | test                                 |
| 6570f0b8-d518-4bf7-bcc4-5e53cc51c911 | centos7.0-7-2-master-clone           |
| cba93690-0748-4eb6-8f64-f5e5c1668997 | centos7.0-7-2-slave01-clone          |
| afbb10f1-2700-49a0-af03-bf7aa9b35149 | centos7.0-7-1-master-clone           |
| 1c935b8b-d862-45fe-bdf9-1310b4989fad | centos7.0-7-1-slave01-clone          |
```

最后，镜像上传成功。

9.5 配置业务系统无缝迁移

在配置完成 Nova 组件后，就可以创建虚机并使用它了。但作为企业级运维人员会多问自己两个问题：如果这个节点有了问题，那么虚机还能不能运行？是否能马上迁移到其他节点？假如一个关键业务在运行时，它的宿主系统突然出现问题，那么所有的业务将被停止，影响用户的使用——想到这里，顿时会让大家感到害怕。如果出现这种情况，我们有两种解决方案：一是建立业务集群，让集群中的虚机运行在不同的节点上（LBaaS）；二是配置云计算环境，实现虚机热迁移功能（Live Migrate）。本节将从虚机热迁移功能方面展开讲解。

首先，需要说明的是虚机热迁移，计算节点必须拥有相同的 CPU 类型。

1. 配置计算节点虚机缓存目录

当我们安装了计算服务，需要运行虚机系统时，虚机都会在宿主计算节点上写入虚机临时文件。而这些临时文件保存的位置，直接决定着虚机是否能进行平滑热迁移。一般情况下，虚机启动后，对虚机添加服务、增加文件等操作都会在宿主机的/var/lib/nova/instances 文件夹内生成由虚机 ID 为编号的临时文件夹，来保存这些数据。那么，在所有的计算节点上使用共享存储挂载到这个目录，就可以满足虚机平滑热迁移的基本条件了。

在分布式存储系统上建立一个资源池 rbd0，然后通过 NFS 功能挂载到计算节点本地：

```
$ sudo mkfs.xfs /dev/rbd0
$ mount /dev/rbd0 /var/lib/nova/instances（最好把磁盘挂载写入/etc/fstab）
```

然后，所有虚机的临时文件都会存放在这个共享存储中。

2. 配置计算节点支持热迁移功能

在计算节点配置中，默认是没有开启迁移功能的，也没有开启虚机资源使用量调整功能。如

何开启，只需要修改/etc/nova/nova.conf 文件即可：

```
[default]
allow_migrate_to_same_host = True
allow_resize_to_same_host = True
```

另外，为了更利于依赖元数据及虚机热迁移（见图 9.15），还需要在配置文件中增加：

```
[libvirt]
live_migration_flag=VIR_MIGRATE_UNDEFINE_SOURCE,VIR_MIGRATE_PEER2PEER,VIR_MIGRATE_
LIVE,VIR_MIGRATE_TUNNELLED
inject_password = false
inject_key = false
inject_partition = -2
hw_disk_discard = unmap
```

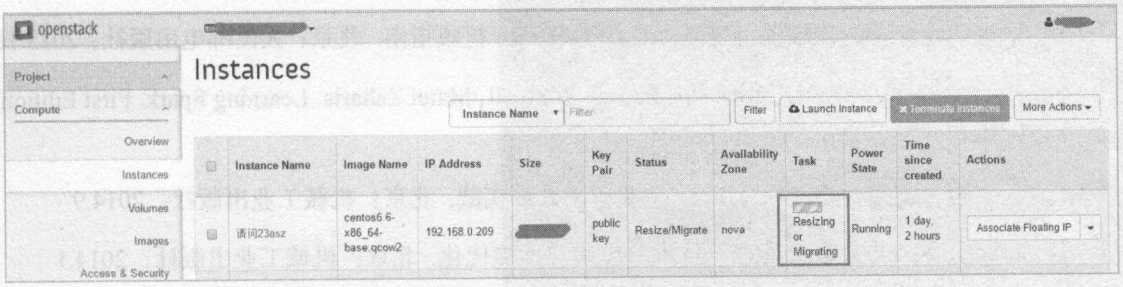


图 9.15　虚机热迁移

至此，在云平台上运行的业务系统就可以实现无缝迁移了。

9.6　本章小结

本章中，我们首先讲解了飞谷云一期、二期项目的系统架构及系统配置。还详细讲解了 OpenStack 组件的实际部署实现，并进行了相应测试；在网络和分布式存储方面，用了更多的篇幅进行讲解，为的是让初学者能够在部署方面了解更多内容和技巧。

其次，根据飞谷云二期项目的实际环境，讲解了在部署、使用云平台上遇到的常用问题并做了阐述。

最后，大家也能发现飞谷云二期项目并没有用到 OpenStack 的所有组件。对于没有使用到的组件，后续我们会根据业务需求逐步进行部署、启用，总结出文档。截至本章内容结束时，已经陆续启用了 Ceilometer、Neutron VLAN、LBaaS 及 Sahara。这些内容，我们会在以后进行补充。

参考文献

【1】 Edward Capriolo，Dean Wampler，Jason Rutberglen 著. 曹坤译. Hive 编程指南. 北京：人民邮电出版社，2013.12

【2】 Lars George 著. 代志远，刘佳，蒋杰译. HBase 权威指南. 北京：人民邮电出版社，2013.10

【3】 Holden Karau，Andy Konwinski，Patrick Wendell，Matei Zaharia. Learning Spark. First Edition. O'Reilly Media，Inc.，February 2015

【4】 马延辉，孟鑫，李立松. HBase 企业应用开发实战. 北京：机械工业出版社，2014.9

【5】 高彦杰. Spark 大数据处理：技术、应用与性能优化. 北京：机械工业出版社，2014.1

【6】 戢友. OpenStack 开源云王者归来. 北京：清华大学出版社，2014.8

【7】 张小斌. OpenStack 企业云平台架构与实践. 北京：电子工业出版社，2015.1

【8】 OpenStack 官方文档：http://docs.OpenStack.org/

【9】 Ceph 官方文档：http://docs.ceph.com/docs/master/

【10】 飞谷云网站：http://www.feiguyun.com/

· 刘未昕 ·

从事IT研发和项目管理工作十余年。使用多种程序设计语言，目前研究方向主要是大数据生态系统，从事金融、数据仓库等领域研发。五年以上IT行业授课、培训经验，并在多所高校担任外聘讲师。

· 吴茂贵 ·

运筹学与控制论专业研究生学历。毕业后主要参与数据仓库、商务智能等方面的项目，期间做过数据处理、数据分析、数据挖掘等工作，行业涉及金融、物流、制造业等。近期主要做复杂数据存储、清理、转换等工作，同时在大数据方面也很有兴趣并投入大量时间和精力，且将持续为之。

Broadview®
www.broadview.com.cn
博文视点·IT出版旗舰品牌
技术凝聚实力·专业创新出版

自己动手做大数据系统

本书从OpenStack云平台搭建、软件部署、需求开发实现到结果展示，以纵向角度讲解了生产性大数据项目上线的整个流程；以完成一个实际项目需求贯穿各章节，讲述了Hadoop生态圈中互联网爬虫技术、Sqoop、Hive、HBase组件协同工作流程，并展示了Spark计算框架、R制图软件和SparkRHive组件的使用方法。本书的一大特色是提供了实际操作环境，用户可以在线登录云平台来动手操作书中的数据和代码，登录网址请参考http://www.feiguyun.com/support。

概要

扫一扫
了解更多

博文视点Broadview

@博文视点Broadview

策划编辑：符隆美
责任编辑：葛　娜
封面设计：侯士卿

上架建议：大数据

ISBN 978-7-121-29586-7

定价：49.00元